

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

**SPC TOOLBOX: A MATLAB
BASED SOFTWARE PACKAGE
FOR SIGNAL ANALYSIS**

by

Dennis W. Brown

September 1995

Thesis Advisor:

Monique P. Fargues

Approved for public release; distribution is unlimited.

DTIC QUALITY INSPECTED

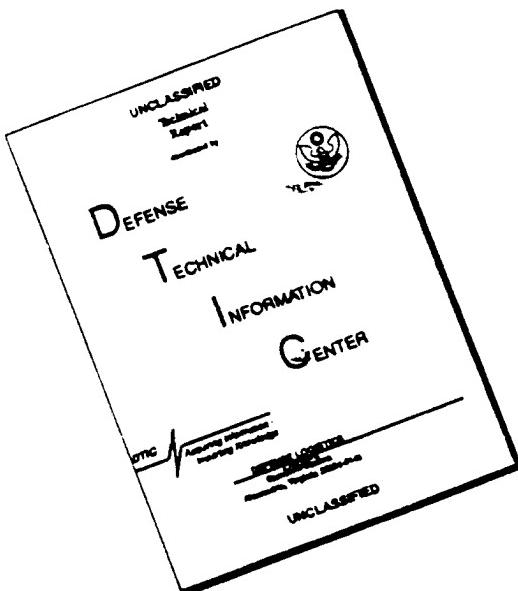
19960208 123

19960208 123

**PAGES 257 THRU 289 ARE NOT MISSING BECAUSE
THE PAGES IN THE REPORT WERE MISNUMBERED
PER ETHEL JOSE (408 656-5097) AT NAVAL POST-
GRADUATE SCHOOL - MONTEREY, CALIF**

MARCH 27, 1996

DISCLAIMER NOTICE



**THIS DOCUMENT IS BEST
QUALITY AVAILABLE. THE COPY
FURNISHED TO DTIC CONTAINED
A SIGNIFICANT NUMBER OF
PAGES WHICH DO NOT
REPRODUCE LEGIBLY.**

REPORT DOCUMENTATION PAGE

*Form Approved
OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time reviewing instructions, searching existing data sources gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)			2. REPORT DATE September 1995		3. REPORT TYPE AND DATES COVERED Master's Thesis		
4. TITLE AND SUBTITLE SPC Toolbox: A MATLAB Based Software Package for Signal Analysis (U)			5. FUNDING NUMBERS				
6. AUTHOR(S) Brown, Dennis W.							
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER				
9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSORING/ MONITORING AGENCY REPORT NUMBER				
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.							
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE				
13. ABSTRACT (Maximum 200 words) This thesis discusses an interactive software developed to complement the teaching of digital signal processing (DSP) concepts in various courses taught by the Electrical and Computer Engineering Department at the Naval Postgraduate School. The SPC software was designed to facilitate the implementation of signal processing concepts learned in the classroom and to illustrate their advantages and drawbacks. The MATLAB® Version 4 environment is used to take advantage of the graphical user interface capabilities. SPC is a window-based, user-friendly tool that allows the user to develop digital filters, analyze signals, and easily design basic signal modeling tools.							
14. SUBJECT TERMS Digital signal process, signal analysis, digital filters.				15. NUMBER OF PAGES 300			
				16. PRICE CODE			
17. SECURITY CLASSIFICATION OF REPORT Unclassified		18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified		19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified		20. LIMITATION OF ABSTRACT UL	

Approved for public release; distribution is unlimited

**SPC TOOLBOX: A MATLAB
BASED SOFTWARE PACKAGE
FOR SIGNAL ANALYSIS**

Dennis W. Brown
Lieutenant, United States Navy
B.S., Kansas State University, 1986

Submitted in partial fulfillment of the
requirements for the degree of

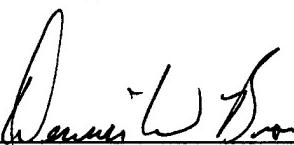
MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL

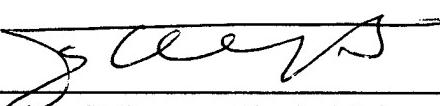
September 1995

Author:

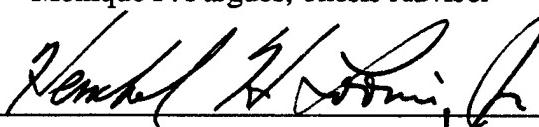


Dennis W. Brown

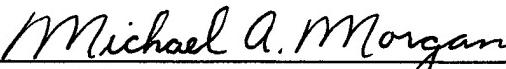
Approved by:



Monique P. Fargues, Thesis Advisor



Herschel H. Loomis, Jr.



Michael A. Morgan

Michael A. Morgan, Chairman,
Department of Electrical and Computer Engineering

ABSTRACT

This thesis discusses an interactive software developed to complement the teaching of digital signal processing (DSP) concepts in various courses taught by the Electrical and Computer Engineering Department at the Naval Postgraduate School. The SPC software was designed to facilitate the implementation of signal processing concepts learned in the classroom and to illustrate their advantages and drawbacks. The MATLAB® Version 4 environment is used to take advantage of the graphical user interface capabilities. SPC is a window-based, user-friendly tool that allows the user to develop digital filters, analyze signals, and easily design basic signal modeling tools.

TABLE OF CONTENTS

I.	INTRODUCTION	1
II.	MATLAB AND THE SPC TOOLBOX	3
A.	MATLAB	3
B.	THE SPC TOOLBOX	4
1.	GUI Application Functions	5
a.	Zoom Tool	5
b.	The Signal Edit Tool	6
c.	The Voice Edit Tool	6
d.	The Signal Filter Tool	7
e.	The Two-Dimensional Spectral Estimation Tool	7
f.	The Three-Dimensional Spectral Estimation Tool	8
g.	The Signal Modeling Tool	9
h.	The Graphical Digital Filter Design Tool	9
2.	Linear Systems Functions	10
3.	Communications Functions	10
4.	Speech Signal Analysis Functions	11
5.	Utility Functions	11
III.	SPC PROGRAMMING PARADIGM	13
IV.	CONCLUSION	17
LIST OF REFERENCES		19
APPENDIX: SPC TOOLBOX USER'S GUIDE		21
INITIAL DISTRIBUTION LIST		291

I. INTRODUCTION

Digital Signal Processing (DSP) is a continuously changing field whose emphasis has increased tremendously in academic institutions in the last ten years. While DSP is quite important in electrical engineering education, the mathematical concepts used can be rather difficult when presented in a classical classroom environment. These concepts become more meaningful when they are actually applied to specific problems. Thus, the last few years have seen numerous innovative proposals to emphasize applications along with theory and to provide a better “hands-on” approach to the teaching of DSP concepts [1,2].

Students studying at the Naval Postgraduate School (NPS) have between twelve months to two and a half years to complete a Master’s Degree depending on their undergraduate degrees. The average student pursuing a Master’s degree at NPS is very different from the average graduate student studying at a civilian U.S. university; he/she is usually older than the “typical” graduate student, has a significant amount of work experience, but has been out of school for as much as eight years. In addition, his or her undergraduate major may be in a discipline other than electrical engineering. As a result of the very diversified backgrounds, NPS students favor hands-on approaches which allows them to better visualize what theoretical concepts mean and how they can be applied in practical applications. Therefore, courses taught in the Digital Signal Processing track in the Electrical and Computer Engineering Department at the Naval Postgraduate School make heavy use of projects and computer-based assignments in an effort to better relate theory and applications.

The initial motivation behind the development was based on the following three observations; 1) similar software is often required to be written by students in various courses (i.e., there is a significant software overlap); 2) software written under these circumstances is usually case specific (i.e., even though the DSP technique implemented may be quite generic, the resulting software has only been thoroughly tested for a very specific application, and may or may not be “valid” for a different application); 3) “home-

coded” software written in early courses is sometimes not properly commented, which significantly reduces the chances of the code being used in another course. Thus, in summary, students potentially “waste” a significant amount of time writing and rewriting similar software for different courses which build on each other. For example, parametric modeling techniques covered in the *Statistical Signal Processing* course are used in *Speech Signal Processing* and *Spectral Estimation* courses taught at NPS. SPC was designed to decrease the need to code some of the basic DSP and communications techniques, and as a result to decrease the time needed for programming. This decrease should allow students to focus more on understanding the methods to be analyzed and to tackle more challenging projects.

SPC is a MATLAB® Version 4 based set of individual tools which are linked together so that different operations can be cascaded. MATLAB version 4.2 and its Signal Processing Toolbox are required to run the software to its full capabilities. SPC functions can be divided into five main groups: time-domain signal analysis, spectral estimation, filtering, signal modeling, communication signal generation, and graphical tools. In addition, SPC makes extensive use of the graphical user interfaces (GUI) offered in MATLAB. SPC uses pull-down menus, push buttons, edit boxes, pop-up menus, check boxes, scroll bars, and drag-and-drop cursor selection. Among the capabilities of this software is the ability for the user to visually design digital filters and to visualize effects of processing signals through these filters, analyzing speech signals, and designing basic parametric AR and ARMA techniques.

II. MATLAB AND THE SPC TOOLBOX

The motivation behind development of the SPC Toolbox is the design a DSP workstation providing a basic set DSP tools allowing researchers and students to:

- Easily examine signals in the time and frequency domains using a wide range of techniques.
- Easily perform fundamental signal processing operations (such as lowpass and bandpass filtering) on signals.
- Easily process signals using more advance techniques.
- Study fundamental concepts of DSP.

Of utmost importance is for the workstation's software to be user-friendly, affordable, and multi-platform, that is, easily transported between different types of computers. Most software considered user-friendly today uses a graphical user interface (often referred to as a GUI or "gooey" interface). GUI interfaces use pull-down and pop-up menus, push buttons, check boxes and other graphical controls which can be activated with a pointing device such as a mouse to select various options and execute commands. The use of a GUI interface relieves the user from having to memorize a large number of often cryptic textual commands - often with a large number of even more cryptic options available. A well-designed GUI interface usually has the added benefit of reducing the amount of time a user spends learning an application and, if a consistent GUI interface is designed across applications, allows the user to more quickly learn a new application having had experience with another application using a similar GUI interface.

A. MATLAB

MATLAB stands for MATrix LABoratory. It is a program providing an interactive environment for solving matrix orientated problems frequently found in mathematics, numerical analysis and engineering. With MATLAB, users are relieved from the low-level programming requirements of programming languages such as C and FORTRAN. As a matter-of-fact, MATLAB was written specifically as an easy-to-use interface to the Fortran LINPACK and EISPACK matrix function libraries. Because it relieves engineers from the

drudgery of traditional programming, MATLAB has found widespread use in university engineering departments and in the engineering industry. The current version of MATLAB is a commercial product of The Mathworks, located in Natick, MA, a company founded in part by Cleve Moler, the original writer of MATLAB. [1]

MATLAB is available from The MathWorks for a wide variety of computer hardware and operating systems, from i386 based personal computers running the MS-DOS operating system to high powered workstations running the UNIX operating system. Beginning with version 4.0, The MathWorks added the capability of building GUI interfaces using pull-down menus, popup menus, push buttons, edit boxes, check boxes, radio buttons and slider controls to MATLAB. This capability, trademarked under the name Handle Graphics® by The Mathworks, is a platform-independent front-end to the underlying platform-dependent graphical user interface for the system MATLAB is executed on. Using Handle Graphics, virtual windows can be created containing controls performing specified actions when selected by the user. This leads to MATLAB programs having the look and feel of programs written using graphical environments such as X-Windows and Microsoft Windows.

B. THE SPC TOOLBOX

The SPC Toolbox was written in the form of a MATLAB toolbox, many of which, are available from The MathWorks. A MATLAB “toolbox” is a collection MATLAB “m-file” scripts and functions which perform various related operations such as statistical analysis or signal processing. Creating the SPC Toolbox required the writing of a large number of “m-files.” Functions written as m-files are interpreted at run-time by MATLAB and can therefore, at time, executed very slowly. To improve performance in the SPC Toolbox, some C language programs were written, as needed, and compiled as callable “mex-file” functions using the MATLAB external interface. All mex-file functions are available in both mex- and m-file form, however, the mex-file forms have a considerable speed improvement over the m-file implementations.

Due to the large number of scripts and functions which required development, the SPC Toolbox the files were grouped by function area with each group placed in a separated sub-directory in the directory organization of the SPC Toolbox. These function groups are described below.

1. GUI Application Functions

This group is the heart of the SPC Toolbox. It consists of a number of time- and frequency-domain signal processing tools along with signal modeling tools and a unique application for the study of digital filters. These tools are briefly introduced below. For more specific information, refer to the Appendix, "The SPC Toolbox User's Manual."

a. *Zoom Tool*

Zoom Tool is a GUI applications that attaches itself to existing plots of vectored data displayed using the MATLAB plot command. This vector could represent a signal or any other form of vectored data. Zoom Tool itself, supplies is the capability of inspecting signals by zooming in and out on portions of the signal, attaching cross-hair cursors to data samples and reading both the x- and y-axis values of the sample from digital readouts, using pairs of cross-hair cursors to make difference measurements between two samples and a scroll capability useful in the search for specific time-domain characteristics in a signal. Zoom Tool is also used by most other GUI applications contained in the SPC Toolbox to provide these functions when the applications contains an axis plot.

Use of Zoom Tool focuses on the use of moving a pair of cross-hair cursors representing the x- and y-axis values for the sample the cursors are currently located over. To position these cursors, Zoom Tool provides number of options for moving them. The simplest method is grabbing the vertical cross-hair cursor with the platform's pointing device (assumed to be a mouse in all further discussions). When grabbed in this manner, the vertical cursor is moved to the desired location by "dragging" the cursor with the mouse while holding down the left mouse button. After the cursor is in the desired position, it is dropped by releasing the mouse button. After dropping the vertical cursor, the horizontal

cursor is moved to corresponding y-value of the data sample the cursor was dropped on top of and digital readouts of the cursor locations are updated. A second method of grabbing a cross-hair cursor is to “click” a spot on the line formed by the plot data vector. In this case, both cursors move to the position of the mouse pointer and as the mouse is moved, the cursors are dragged and their current positions are continually updated on the digital readouts. The cross-hair cursors can also be positioned by pressing push buttons which move the vertical cursor to the next left or right sample and push button which move the vertical cursor to the next extrema (up or down “peak”).

b. The Signal Edit Tool

The Signal Edit Tool allows the user to edit signals by “cut and pasting” portions of the signal using the platforms pointing device. For example, the user can mark a section of signal for which further processing is desired. The outlying ends of the signal can then be “cropped” and the signal saved or additionally processed by the Signal Filter Tool (or a similar tool). Removing undesired sections of signals before further process has the advantage of eliminating the time and overhead of processing irrelevant data. In addition to editing a signal, the Signal Edit Tool can be used simply as a means of viewing the time-domain of a signal, allowing the user to inspect portions of a signal by zooming in and out on desired portions. This is basically the same functionality Zoom Tool provides only the y-axis cursors and digital readouts are not displayed. Pull-down menus also allow the user to “play” the signal over the platform’s audio output.

c. The Voice Edit Tool

The Voice Edit Tool is an enhanced version of the Signal Edit Tool, optimized for use with speech signals. It provides most of the functionality of the Signal Edit Tool while displaying the short-time energy, magnitude and zero-crossing information curves of the signal. These curves aid the user in picking voiced and unvoiced phonemes contained in the speech. Once individual phonemes “picked” out of a speech signal, they can be individually modeled using the Signal Modeling Tool as a means of synthesizing speech.

d. The Signal Filter Tool

The Signal Filter Tool provides a means for the user to quickly design and then apply digital lowpass, highpass, bandpass, and stopband filters to signal. This tool supports both Finite Impulse Response (FIR) filters and Infinite Impulse Response Filters (IIR). The IIR filters are sub-divided into digital prototypes of analog Butterworth, Chebychev and elliptical filters. During filter design, both the spectrum of the signal being filtered and the transfer function of the currently designed filter are display on the same frequency-domain plot. As filter parameters (e.g. type, order, bandpass ripple, etc.) are change, the transfer curve changes to reflect the new transfer curve. This ability allows the user to “fine tune” a filter design to remove the unwanted frequencies from a signal. Part of this fine tune capability allows the user to select the desired cutoff frequencies using the mouse or by entering a value for the cutoff frequency. After a filter with the desired characteristics has been designed and applied to the signal, the spectrum of the filtered signal is computed and displayed along with both the transfer function curve and the “before” spectrum. Displaying these three curves on the same plot allows the user to see just how effective the filter was at removing unwanted frequencies.

In addition to the frequency-domain display, the Signal Filter Tool can display the signal in the time-domain in an additional window. This display uses Zoom Tool to provide all the capabilities it supplies. Additionally, menus are available to play the signal over the platform’s audio output and to launch additional GUI tools containing all or a portion of the time-domain signal displayed.

e. The Two-Dimensional Spectral Estimation Tool

The Two-Dimensional (2D) Spectral Estimation Tools provides a means of estimating the frequency versus magnitude spectral estimate of a signal. A large number of spectral estimation techniques are available including classical, parametric and subspace methods. Classical methods include averaged periodograms (both Bartlett and Welch procedures), the averaged frequency bin (Daniell) periodogram and the correlogram

(Blackman-Tukey) methods. Parametric methods include the auto-correlation method, covariance method, modified covariance method and Burg methods of auto-regressive (AR) model estimates. Moving-average (MA) model estimation methods provided are the Prony, Shank, and Durbin procedures. Any combination of the above AR and MA method can be combined to produce an auto-regressive, moving-average model (ARMA) estimate. In addition to these AR, MA, and ARMA estimates, an additional MA method is provided using linear prediction. Subspace methods include the minimum-variance (maximum-likelihood), Pisarenko harmonic decomposition, Multiple Signal Classification (MUSIC) and eigenvalue weighted MUSIC algorithms are provided. Multiple estimates using different method can be displayed on the same plot for direct comparison of the spectral estimates.

f. The Three-Dimensional Spectral Estimation Tool

The Three-Dimensional (3D) Spectral Estimation Tool provides a means of displaying the spectral surface formed by the time versus frequency versus magnitude spectral estimate using the fast Fourier transform (FFT). In the initial window, a frequency versus magnitude spectral estimate for the signal is provided along with Zoom Tool allowing spectral features to be measures. A pull-down menu allows the choice of six surface plots provided using MATLAB. These are surface, lighted surface, mesh, lighted mesh, waterfall and pseudo-color (spectrogram) plots. These plots are displayed in color on supported hardware and the user can select the color map and shading method applied to the three dimensional surface. Choice of color map and shading are important in defining the visibility of certain spectral features. After a spectral surface has been generated, it is displayed in a separate window containing scroll bars which allow the surface to be rotated in three dimensions to obtain the best possible view of the spectral surface. Additional controls allows change the color map and shading methods without regenerating the surface. Any number of 3D surfaces can be generated at any one time (limited only by

platform capabilities) allowing direct comparisons using different surfaces, FFT lengths, frame lengths, adjacent frame overlaps and color map schemes.

g. The Signal Modeling Tool

The Signal Modeling Tool provides a means to model all or a portion of a signal. Currently, all the AR, MA, and ARMA models discussed under the 2D Spectral Estimation Tool are available. In designing a model, graphical controls are used to set the modeling method and order. The user can also use cursors to define a portion of a signal to be used in generating the model parameters. An additional set of cursors allows the user to define the length of the model to be generated. In the case of modeling a voiced speech phoneme, this feature allows the user to define a section of the signal containing several pitch periods to be used in generating the model while only generating a model the length of one pitch period. A chain of these individual pitch periods can be generated and displayed, or played over the workstation audio output to see how accurately the model synthesizes the phoneme modeled.

h. The Graphical Digital Filter Design Tool

The Graphical Digital Filter Design Tool provides a means of designing digital filters by placement of poles and zero directly on a plot of the complex plane. As each pole or zero is added to the complex plane, the filter transfer function representing the magnitude and phase along the unit circle is displayed in additional axis. Placement of poles and zeros can also be accomplished by entering pole/zeros locations directly in either rectangular or polar form. Complex-conjugate pairs of poles and zeros are placed by simply placing one pole or zero in the complex-conjugate pair. The other pole or zero is added automatically. In addition to placing additional poles and zeros with the mouse, existing poles and zeros can be grabbed and dragged to new locations with the mouse. The mouse can also be used to delete existing poles and zeros. By being able to easily place and move poles and zeros, it is hoped the student can gain an intuitive feel for where poles and zeros have to be placed to produce a filter with a desired frequency response. The Graphical Filter

Design Tool is the only GUI applications in the SPC Toolbox written exclusively for educational purposes. However, filters designed with the Graphical Filter Design tool can be used to actually filter digitized signal.

2. Linear Systems Functions

The Linear Systems set of functions deal with generating AR, MA, and ARMA modeling, spectral estimation, adaptive filtering and other similar linear systems related technique. All functions in this group are usable as command line functions in MATLAB. A number of functions in this group are used by the Signal Filtering, Signal Modeling, and the 2D Spectral Estimation Tools. The GUI applications, in essence, act as GUI front-ends for the functions they call from this group.

3. Communications Functions

The Communications group of functions consists of a number of routines which generate several baseband signals (i.e. sinusoids, square waveforms, triangular waveforms, digital unipolar signals, etc) and passband communications signals (on-off keyed (OOK), binary phase-shift keyed (BPSK), and binary frequency-shift keyed (BFSK) signals). AM modulation and demodulation routines are also provided. The main purpose of the inclusion of this group of functions in the SPC Toolbox is as a means of generating signal to be used for study with some of the GUI applications. For example, the spectrum of various communications signals can be studied by generating a sample set of these signal and then using the 2D Spectral Estimation tool to study not only the signal spectrums but also study the performance of the various spectrum estimations techniques when applied to specific types of signals. Note that no functions from this group are called directly by any GUI applications. Signal generation must be done using these functions from the MATLAB command prompt.

4. Speech Signal Analysis Functions

The Speech Signal Analyses group of functions produce estimates of the short-time energy, magnitude, and zero-crossing rates in speech signals. These functions are used by the Voice Edit Tool to produce the short-time information curves used as an aid in parsing phonemes from speech signals. Additionally, functions for smoothing these curves are provided in this group. Just like the Linear Systems group of functions, functions in the Speech Signal Analysis group can be called from the MATLAB command prompt.

5. Utility Functions

The Utility group of functions contain miscellaneous functions which do not fit into any of the above function categories. These deal with data file input/output, framing vectored data, and plot generating. Functions in this group all called by a number of the GUI applications and can also be called directly from the command prompt.

III. SPC PROGRAMMING PARADIGM

The development of the SPC toolbox included the development of a new MATLAB programming paradigm. This paradigm, referred to thereafter as the “Runtime” paradigm, was necessary to achieve the high degree of integration of separate SPC GUI tools and the ability to run multiple copies of the same tool.

Programming a GUI interface in MATLAB requires the ability to reference “handles” of GUI controls. A “handle” is best described as a pointer, and is very similar to the pointers found in C⁺⁺. Referencing handles is necessary for finding and/or setting the current state of a control. Two program paradigms for keeping track of handles are provided by the MathWorks (TMW) [4]. The first TMW paradigm makes use of declared global variables for handle storage. The second one involves creating a vector of control handles and storing the resulting vector in the UserData property of the figure window the GUI interface is created in. This vector can be retrieved, when necessary, giving the programmer access to the desired handle. These paradigms are referred to as the “Global Variable” paradigm and “UserData” programming paradigm respectively. Both of these paradigms were deemed inadequate in developing the SPC toolbox for the following reasons:

- Use of the Global Variable paradigm prevents running multiple copies of a tool since starting a second tool will destroy the stored handles to the first tool.
- Although proper use of the UserData paradigm allows for multiple copies of a tool to be executed, remembering the index in the handle vector is cumbersome for the programmer. It should be noted however, that both paradigms do have a speed advantage over the Runtime paradigm about to be presented [3], and both are used in the SPC Toolbox in certain situations requiring this advantage.

Understanding the Runtime paradigm first requires an understanding of MATLAB Handle Graphic objects. All graphical objects in MATLAB, from figure windows to push button controls, are objects having parent-child (tree) structured relationships. Note that all figure windows have the same parent, the non-displayable “root” object, whose handle is always zero. In addition to object handles, understanding object properties is required for

the Runtime paradigm. Object properties define the size and appearance of controls and the response of controls and other graphic objects to various stimuli (key strokes, mouse actions). Object properties are set when created or by using the set command. For example, the partial command

```
uicontrol('Style', 'pushbutton','String','OK',....)
```

is a push button control labeled “OK”. Not all properties have to be set explicitly by the user. In the above example, the property “Type” was automatically set to the value “uicontrol” by the UICONTROL command. The GET command can be used to retrieve property values which can then be tested. For example, the commands

```
label = get(handle,'String')
if strcmp(label,'OK'),
    ... action ...
end
```

retrieves the value stored in the String property and perform “action” if the String property is equal to OK.

Two properties, Parent and Children, are key to the Runtime paradigm. As one would expect, the Parent property contains the handle of the parent object and the Children property contains a vector of handles to all the children of an object. By retrieving handles to the parent and children objects, the object tree can be traversed either up, down, or across. At this point, the Runtime paradigm can be defined. The basic idea behind this new paradigm is that any object, having a set of known object properties, can be found by traversing the object tree formed by the Parent and Children properties until the object has been found. That is, object handles can be found at Runtime, hereby, eliminating the need to store them.

In the SPC Toolbox, the “find” functions support the Runtime programming paradigm. For instance, the command

```
h = findpush(gcf, 'OK')
```

would retrieve the handle of the push button control in the previous example. Beginning with MATLAB version 4.2, The MathWorks included a built-in command to directly

support the Runtime programming paradigm. The find object command (FINDOBJ) would be used as

```
h = findobj(gcf, 'Type', 'uicontrol', 'Style', 'pushbutton',
'String', 'OK')"
```

in place of FINDPUSH.

IV. CONCLUSION

The SPC Toolbox is a user-friendly, interactive MATLAB-based toolbox designed to assist students in the application of DSP concepts learned in the classroom. Its graphical capabilities facilitate the analysis of data and decrease the programming required for students. The wide-variety of tools available in the SPC Toolbox make its use easily integrated into the classroom. A detailed presentation of the software may be found in the following appendix which contains the SPC Toolbox user's guide.

LIST OF REFERENCES

1. Special Session on Signal Processing Education, IEEE ICASSP-94 Proceedings, Vol. 6, April 1994.
2. Special Session on Signal Processing Education, IEEE ICASSP-93 Proceedings, Vol. 1, April 1993.
3. Building a Graphical User Interface, The MathWorks Inc., 1993.

APPENDIX: SPC TOOLBOX USER'S GUIDE

SPC Toolbox

An Interactive Matlab Package for
Signal Modeling and Analysis
and Communications
(with Speech Analysis
and
Linear Systems Modeling)

LT Dennis W. Brown and Dr. Monique P. Fargues
Department of Electrical and
Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5121
September 1995

Table of Contents

Introduction	1
Signal Edit Tool Tutorial	7
Signal Filter Tool Tutorial	11
BPSK Modulation/Demodulation Tutorial	15
Tool Reference.....	21
aledsgn	23
armadsgn	29
gfilterd	37
sigedit	54
sigfilt	67
sigmodel	82
spect2d	99
spect3d	117
v3dtool	128
voicedit	134
zoomtool	140
zoomcust	147
zoomplay	148
zoomprog	149
Communications Function Reference.....	151
antpodal	153
bfsk, bfskmsg	154
bpsk, bpskmsg	155
corrdmod	156
coswave, sinwave	158
dsblc	159
dsbsc	160
envelope	161
lrs	162
modindex	163
ook, ookmsg	164
proddmod	165
sawwave	167
setsnr	169
sqwave	172
str2masc, masc2str	174
triwave	175
unipolar	177
Linear Function Reference.....	179
ar_burg	181
ar_corr	182
ar_covar	183
ar_durbn	184
ar_levin	185

ar_mdcov	186
ar_prony	187
ar_shank	188
avgpergm	189
blacktuk	191
daniell	193
freqeig	195
lmsale	196
ls_svd	197
ls_whopf	198
minphase, maxphase	199
minvarsp	201
musicsp, musicspw	203
normaleq	205
rxxcorr, rxxcovar, rxxmdcov	206
showeig	209
Speech Function Reference	211
avsmooth, mdsmooth	213
loadau, saveau	215
loadvoc, savevoc	216
loadwav, savewav	217
readsig	219
sp_steng, sp_stmag, sp_stzcr	220
SSPI Function Reference	223
loadsspi,savesspi	225
Utility Function Reference.....	227
fedit	229
framdata	230
ld8bit, ld16bit, ld32bit, sv8bit, sv16bit, sv32bit	231
lperigrm	232
plottime	234
wperigrm	235
xpilabel, ypilabel	237
Programming Function Reference	239
crsrmake, crsrmove, crsrlc, crsron, crsroff, crsrdel	243
findaxes	245
findchkb, findpush, findrrio, findslid, finduitx	246
findedit, findpopu	247
findfig	248
findfram	249
findline	250
findmenu	251
findtext	252
mufirwin, gtfirwin	253
radiogrp	254
togmenu, getcheck, togcheck	255

zoomadd, zoomrep, zoomdel	257
---------------------------------	-----

Preface

The SPC (Signal Processing & Communications) software package is the result of work originated in Spring 1993 at the Naval Postgraduate School as a class project in speech processing. The project developed an interactive, user-friendly, tool to analyze speech signals using MATLAB. Then, as with any project left without a fixed deadline, it grew to include more sophisticated analysis tools such as various filtering techniques, Autoregressive (AR) and linear Auto Regressive-Moving Average (ARMA) modeling methods. Finally, it was expanded to include basic communication tools.

The final product is a software package designed to provide the user with a series of data manipulation tools which use MATLAB v.4 graphical user interface controls. SPC can be used in the classroom to illustrate and to reinforce basic concepts in digital signal processing and communications. It frees the user from having to write and debug his/her own code and gives him/her more time to understand the advantages and drawbacks of each technique included in the package. It can also be used as a basic analysis and modeling tool for research in Signal Processing. SPC is well suited to reinforce basic concepts presented in the following courses offered at the Naval Post-graduate School:

EC4410: Speech Processing,
EC4420: Modern Spectral Estimation,
EC3420: Statistical Digital Signal Processing,
EC3400: Digital Signal Processing,
EC2500: Communication Theory,
EC2400: Discrete Systems.

We hope that users will find this package useful, and we welcome any comments and suggestions regarding this software at:

dwbrown@access.digex.net
or
fargues@ece.nps.navy.mil.

An early version of the SPC Toolbox was previously released as the following Naval Postgraduate School Technical Report.

Brown, Dennis W. and Monique P. Fargues, SPC Toolbox: An Interactive MATLAB Package for Signal Modeling and Analysis, and Communications, Technical Report No. NPSEC-93-017, Naval Postgraduate School, Monterey, CA, October 15, 1993.

INTRODUCTION

The SPC Toolbox provides a number of commands implementing a variety of digital signal processing and communication techniques. All of the routines are implemented using MATLAB (version 4.0 to 4.2) m-files with some functions implemented as both m-files and mex-files (written in the C programming language). Most of the commands are stand-alone or rely on other SPC Toolbox routines. Some commands however, rely on commands from the MATLAB *Signal Processing Toolbox* available from The MathWorks. The functions that rely on the *Signal Processing Toolbox* are routines that implement FIR windows, FIR filter design, and digital filter design of analog IIR filter prototypes.

The SPC Toolbox commands are grouped into the following areas:

- Graphical Interfaced DSP Tools.
- Communications including modulation/demodulation.
- Linear systems including AR/MA/ARMA modeling and adaptive filtering.
- Speech signal processing.
- Formatted signal data file loading and saving.

A number of graphics-based GUI tools provide interactive, graphical environments offering relief from the MATLAB command line prompt. Some of these tools essentially act as “front-ends” to other functions. Others provide interactive modeling facilities (i.e. AR/MA/ARMA modeling). Still, others, provide graphical environments to perform tasks not readily available or easily performed using MATLAB commands (i.e. visual editing of signal). The following tools are some of those provided in the SPC Toolbox:

- Signal Edit Tool (a visual “cut and paste” signal editor, `sigedit`).
- Speech Signal Edit Tool (a visual “cut and paste” signal editor for speech, `voicedit`).
- Signal Filtering Tool (interactive filtering, `vectfilt`).
- Signal Modeling Tool (interactive modeling, `vectarma`).
- 2D Spectral Estimation Tool (frequency/magnitude spectral estimation, `spect2d`).
- 3D Spectral Estimation Tool (time/frequency/magnitude spectral estimation, `spect3d`).
- Spectrum Scope Tool (signal playback display of the time and frequency domains, `spscope`)

SPC Toolbox Conventions

The following conventions are followed by all commands in the SPC Toolbox:

- All commands have been tested in MATLAB versions 4.1 and 4.2, running on Sun workstations or on personal computers running Microsoft Windows MATLAB versions 4.0 and 4.2 (version 4.1 was never released for MS-Windows). Although untested, SPC Toolbox commands should work with no problems on other platforms running MATLAB versions 4.1 or 4.2. The exceptions are versions of MATLAB which do not have the undocumented `uicontrol` property `Enable` which is used throughout the GUI tools, and the undocumented function `gs_modal_dialog` used by the GUI tools for displaying error messages and contained only in the file `spcprog/spcwarn.m`. The `spcwarn` function should auto-detect the absence of the

`gs_modal_dialog` command and use a work-around method of displaying error messages. See the `spcwarn` command in the programming tools reference section of this document for more information.

- Except for tools, no commands are interactive (i.e. request input from the user via prompts).
- All commands taking signals as input arguments accept either Nx1 or 1xN vectors.
- Vectors are returned as Nx1 vectors (except functions returning vectors representing polynomials which return 1xN vectors following the MATLAB convention for polynomials).
- Arguments returned after an error occurs are returned as empty vectors. This allows error checking by user implemented routines with the following code:

```
y = bpsk(50,10240);
if isempty(y),
    error('An error occurred generating a BPSK signal.');
end;
```

- The default sampling frequency for all function which take a sampling frequency as an input argument is 8192 Hz. In GUI tools which allow playing signals over the workstation audio output, the sampling frequency on the tool's sampling frequency popup menu is used to send the signal to the audio output. If the audio output does not support a sampling frequency, the default sampling frequency used is the same as that of the MATLAB `sound` command.
- On-line help is available by typing `help command_name` at the MATLAB prompt. A description of SPC Toolbox commands can be viewed by typing

```
help spctools
help spcgui
help spccomms
help spcline
help spcspch
help spcutil
help spcspi
help spcprog
```

- In the documentation, commands are printed in a sans-serif font and variable names, file names and directory names are printed in *italics*. Commands in the table of contents are shown in capital letters. All examples should work as shown if the toolbox is properly installed.

Customizing

Most colors within GUI tools can be customized to meet individual preferences by editing the file `spcolors.m`. On workstations, individual users can copy the `spcolors.m` file from the `spctools` directory and modify it to meet their needs. To make individual copies of `spcolors.m` take precedence over the default `spcolors.m`, the only requirement is that the individual's copy of `spcolors.m` is reachable in the `matlabpath` before the `spctools` directory. This can be accomplished by keeping a copy of `spcolors.m` in the working directory or by prepending the `matlabpath` with the directory containing the individual's copy of `spcolors.m`. See the MATLAB *User's Manual* for information on the `matlabpath` variable.

In addition to colors, the default size and location of most tools may be specified in the *spcolors.m* file. Simply “uncomment” these lines according to the instructions in the *spcolors.m* file and replace the parameters for the location and size with the user’s preference. The default screen sizes are nine tenths of the screen for screens less than 800 pixels wide, eight tenths of the screen for screens from 800 to less than 1024 pixels wide and seven tenths of the screen for screens 1024 pixels wide and larger. The default size and location for child graphic windows, in general, cannot be specified in *spcolors.m*.

The file *spbandw.m* contains the default colors for black and white screens. To use this file, copy or rename the file to *spcolors.m* and follow the instructions above. The *spbandw.m* configuration file expects the MATLAB *whitebg* command to be executed before running any GUI tools.

Error Messages

A considerable amount of error checking is performed in the SPC Toolbox routines in an attempt to avoid disasters. All error messages generated by SPC Toolbox routines have the format:

```
function_name: error message
```

If you receive an error message that does not have the name of a SPC Toolbox routine, the error message was generated by MATLAB or a command from some other toolbox. Currently, the SPC Toolbox relies only on version 4.1 (Sun) and 4.0 (MS-Windows) MATLAB commands and a few commands from the MATLAB *Signal Processing Toolbox*. In a few cases, commands from version 4.2 will be used automatically in lieu of earlier commands where the 4.2 commands are faster.

If you receive an error while executing a SPC Toolbox command, please send email to *fargues@ece.nps.navy.mil*. Include a description of the activities in progress when the error was received and a copy of the error message itself. If possible, cut the command and error message right from the shell used by MATLAB and paste it into email text using the X Windows (or MS-Windows) cut-and-paste facilities.

Interaction with X-Windows

The SPC Toolbox graphical tools push the limits on MATLAB’s graphical capabilities on X-Window hosted (UNIX) machines. It is not uncommon to receive bus errors, segmentation violations, or IO Error error messages which contain instructions from the Mathworks for contacting them about the error. If you can reproduce the error message, please contact The Mathworks (as instructed in the error message) and not the Naval Postgraduate School. These errors are caused by bugs in MATLAB, not the SPC Toolbox. The occurrence of these errors occur less often when using version 4.2.

Installation Under MATLAB for Microsoft Windows

The following steps are required to install the SPC Toolbox on a personal computer with MATLAB running under Microsoft Windows:

From an uncompressed distribution disk:

1. To load the SPC Toolbox onto a PC, from the MS-DOS prompt change to the

- ..\MATLAB\TOOLBOX sub-directory.
2. Place the floppy into a drive and execute

```
xcopy a:\spctools spctools /s
```

from the MS-DOS prompt. If you have not loaded the SPC Toolbox previously (e.g. the \MATLAB\TOOLBOX\SPCTOOLS directory does not already exist), you'll be asked if *SPCTOOLS* is a directory or a filename. Respond that it is a directory. After extensive disk activity, SPC Tools will be loaded onto your hard drive. The bulk of the SPC Toolbox files will be located in subdirectories below the *SPCTOOLS* directory.

From a compressed distribution file:

1. Create an *SPCTOOLS* sub-directory in the \MATLAB\TOOLBOX directory.
2. Copy the files spctools.zip the this directory and uncompress the file using the command

```
pkunzip -d spctools.zip
```

which will uncompress the file, creating a number of subdirectories containing all the SPC Toolbox files.

Using either method:

3. Edit the *MATLABRC.M* file located in the ..\MATLAB directory. Find the section similar to the following example *MATLABRC.M* file and insert the lines:

```
....  
'drive_letter:\matlab\toolbox\spctools;','...  
'drive_letter:\matlab\toolbox\spctools\spcgui;','...  
'drive_letter:\matlab\toolbox\spctools\spcline;','...  
'drive_letter:\matlab\toolbox\spctools\spcprog;','...  
'drive_letter:\matlab\toolbox\spctools\spcspch;','...  
'drive_letter:\matlab\toolbox\spctools\spccomms;','...  
'drive_letter:\matlab\toolbox\spctools\spcutil;','...  
'drive_letter:\matlab\toolbox\spctools\spcspi;','...  
....
```

The location of these lines can be critical if any command in any other toolboxes has the same name as a SPC Toolbox command. Some care has been taken to name SPC Toolbox commands differently from those used in the MATLAB *Signal Processing Toolbox*, *Control Systems Toolbox* or *System Identification Toolbox*. To ensure SPC Toolbox commands take precedence over any other toolbox commands of the same name, enter the above lines near the top of the *matlabpath* specification. To ensure SPC Toolbox commands do not take precedence, enter the above lines near or at the bottom of the *matlabpath* specification (as shown below).

```
% Setup the MATLAB search path.  
matlabpath(['...  
'e:\MATLAB','...  
'e:\MATLAB\toolbox\matlab\general;','...  
'e:\MATLAB\toolbox\matlab\demos;','...  
....  
'e:\matlab\toolbox\signal','...  
....
```

```
'e:\matlab\toolbox\ident',...
'e:\matlab\toolbox\control',...
'e:\matlab\toolbox\spctools',...
...
'e:\matlab\myprogs',...
]);

```

Installation on Sun Workstations/Networks

1. Create a *spctools* sub-directory in the *matlab/toolbox* directory.
2. Change to the *spctools* sub-directory using the UNIX *cd* command.
3. Copy the compressed distribution tar file *spctools.tar.Z* to the new *spctools* directory.
4. Uncompress the distribution file with the command “uncompress *spctools.tar.Z*”.
5. Untar the distribution file with the command “tar xvf *spctools.tar*”.
6. Add the *spctools* directory to the *matlabpath* or add a link in an existing directory in the *matlabpath* to the *spcpath.m* file located in the *spctools* directory.
7. Either add the *spctools* directories to the *matlabpath* or ensure the *spcpath.m* file is configured and accessible for your installation. With the *spcpath* command, users are required to execute *spcpath* before gaining access to the complete SPC Toolbox. This option avoids unknowing users from executing a SPC Toolbox command when actually trying to execute a command by the same name in another toolbox.

Disclaimer

This software package is made available as a service to the academic community. It is not a depository of Mathworks approved software. The authors make no warranty of any kind, expressed or implied, with regard to these programs or the documentation contained in this manual. Use of these programs is at the users' own risk and the authors disclaim all liability for injury, damage, and losses that may result from their use. Losses include, but are not limited to, loss of data or data being rendered inaccurate, or losses sustained by third parties for a failure of the software to operate. The U.S. Government assumes no responsibility for the information provided. MATLAB is a trademark of The Mathworks. Microsoft Windows is a trademark of Microsoft Incorporated. Sun is a trademark of Sun Microsystems. Soundblaster is a trademark of Creative Labs, Inc.

Introduction

TUTORIAL

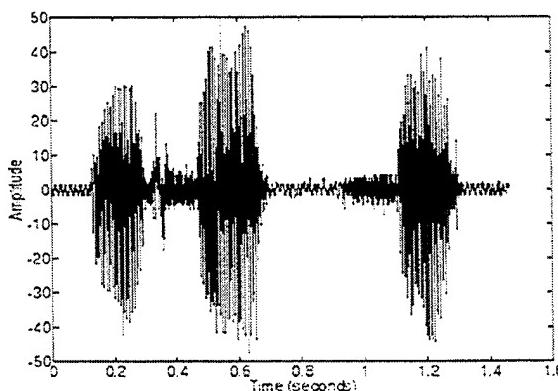
Signal Edit Tool

This tutorial introduces the basic operation of the Signal Edit Tool (*sigedit*). The Signal Edit Tool provides an interactive, graphical environment within MATLAB to edit signals (naturally). Note, *sigedit* will work equally well with any MATLAB vector, not just those representing signals. In this tutorial, we will use *sigedit* to rearrange the words in a speech signal.

A file named *seatsit.voc* is contained in the distribution and located in the *spctools* directory. This file contains the words “the seat, sit” as spoken by the author. It was recorded on a PC using a microphone equipped Soundblaster audio board. The file is stored in the Soundblaster Creative Voice file format (*.voc). To load the file into the MATLAB workspace, the *loadvoc* command from the SPC Toolbox is used:

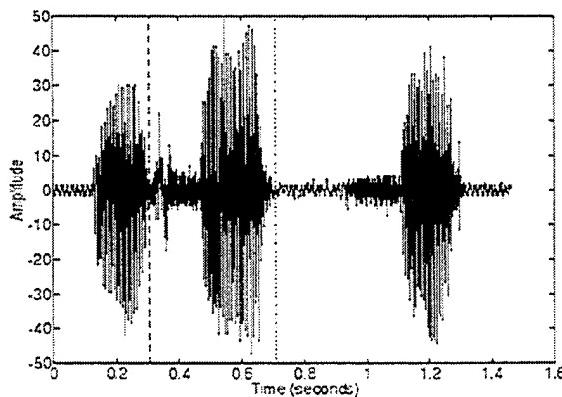
```
seatsit = loadvoc('seatsit'); % insert pathname as appropriate
```

Start the Signal Edit Tool with the command *sigedit(seatsit,8000)*. This command loads the vector *seatsit* into the *sigedit* tool from the MATLAB workspace. Note the command *sigedit('seatsit.voc')* could also have been used here. Once loaded, the plot in the *sigedit* tool window will look like the following.

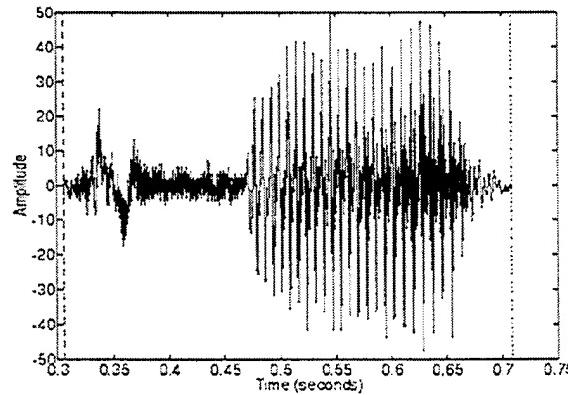


Concerning this signal, note that a 60 Hz interference can be seen during periods of silence. Second, note the /s section of the word “seat” contains a large, low-frequency transient. Third, note that this signal was sampled at 8000 Hz and can be sent to the audio output of a Sun workstation (or soundcard equipped PC) by using the Play pull-down menu.

In this tutorial, we will cut the word “seat” and place it in front of the word “the”. Begin by surrounding the word “seat” with the cursors as shown below. Note that *zoomtool* is part of *sigedit*’s interface. These cursors can be moved by dragging with the mouse pointer or by entering values into the cursor position edit boxes. Once you have placed the cursors, select Play-Cursors pull-down menu. This menu item sends the signal located between the cursors to the audio output and is useful to ensure you have only the word “seat” surrounded.

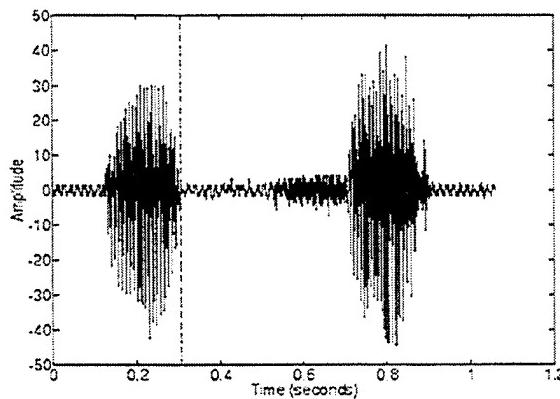


With the word "seat" surrounded by the cursor, click on the push button labeled "><" under the "XY" label on the right side of the axis. This push button is the zoomtool control for zooming in on the area between the vertical cursors. After pressing the zoom push button, the plot in sigedit should look similar to the following.

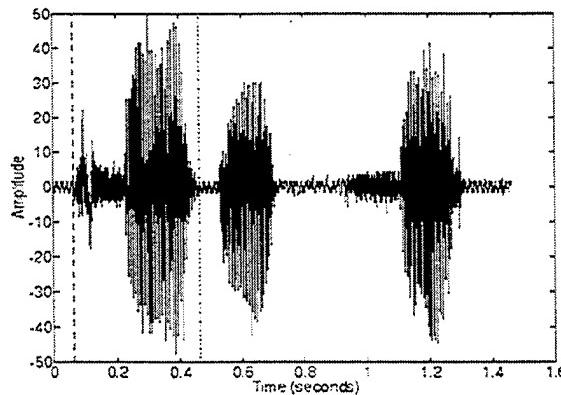


Now, the low frequency transient can be seen more clearly. If desired, reposition the cursors around the low-frequency transient and select the "><" push button once again to zoom in further. Once you are satisfied with the capabilities the zoom-in feature, select the "<>" push button to zoom out to previous zoom in positions. At any time, the "[]" push button can be used to the entire signal. After becoming familiar with the zoom capabilities, zoom out to display the whole signal and then reposition the markers to surround the word "seat" once again.

In reordering the words, we cut the word "seat" and then paste it into the signal before the word "the." With the markers surrounding the word "seat," select the Edit-Cut pull-down menu with the left mouse button. Selecting the Edit-Cut pull-down menu removes the portion of the signal between the two vertical cursors and stores it in a cut-and-paste buffer. After cutting the signal, sigedit collapses the signal and redisplays it with cursor 1 (---) located on the first sample before the cut and with cursor 2 (-.-) located on the first sample after the cut. As long as the cursors remain in these positions, the cut may be undone by selecting the cut push button again.



The remaining step consists in pasting the word “seat” into the signal before the word “the.” Select the Edit-Paste pull-down menu with the left mouse button. After selecting Edit-Paste, the cursor changes into a cross-hair and the plot title changes to “*Mark insertion point with cursor.*” Move the cursor to the period of silence before the word “the” and press the left mouse button. The word “seat” is now the first word in the signal. Selecting the Play-Full pull-down menu will confirm this fact.



See the sigedit reference page for further discussion on the sigedit tool. This tutorial applies equally well to the Voice Edit Tool (voicedit).

TUTORIAL

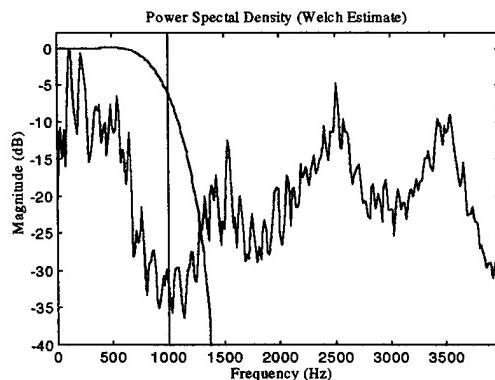
Signal Filter Tool

This tutorial introduces the basic operation of the Signal Filter Tool (`sigfilt`). The Signal Filter Tool provides a graphical, interactive environment to design filters and to apply the filter to a signal. To aide in designing a filter that exactly meets the need of the problem at hand, `sigfilt` displays both the filter transfer function and the spectrum of the signal to be filtered simultaneously. This capability allows the user to select the filter cutoff frequencies by placing the cutoff frequency cursors directly around the spectral features of the signal to be either removed or saved.

In this tutorial, we design a Chebychev Type II bandpass filter to study the formant frequencies contained in a speech signal. Specifically, we will band-pass filter the mid-band frequencies and play the result to hear the contribution of this frequency band to the speech. A file named `seatsit.voc` is contained in the distribution and located in the `spctools` directory. To load the file into the MATLAB workspace, the `loadvoc` command from the SPC Toolbox is used:

```
seatsit = loadvoc('seatsit'); % insert pathname as appropriate
```

Start the Signal Filter Tool with the command `sigfilt(seatsit,8000)` where the sampling frequency is equal to 8000 Hz. This command loads the vector `seatsit` into the `sigfilt` tool from the MATLAB workspace. Note the command `sigfilt('seatsit.voc')` could also have been used here. Once loaded, the plot in the `sigfilt` tool window will look like the following.

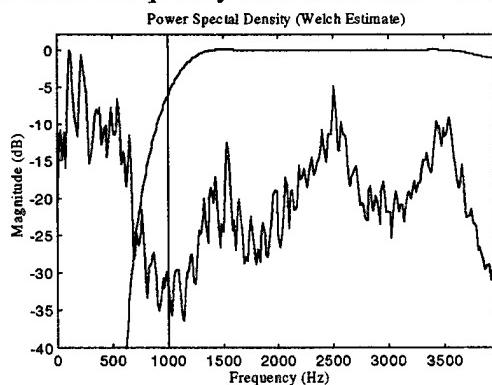


Note: Both time and frequency domains are available in the `sigfilt` tool. Checking the Filter-Display Signal menu item will display the time-domain signal in a separate figure window.

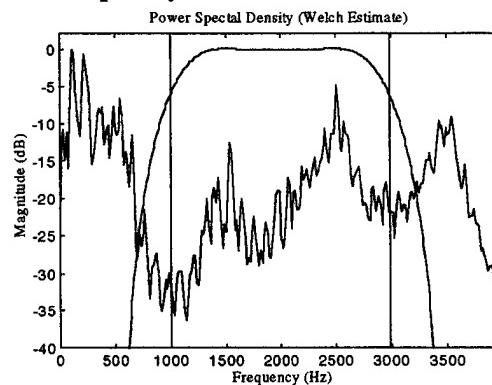
The spectrum display is an average periodogram with non-overlapping frames using a 1024-point FFT. See the Matlab Signal Processing Toolbox `spectrum` command for further information. Only the positive half of the frequency spectrum is displayed and the frequency resolution is half the sampling frequency divided by 512. In the spectral plot, both linear and logarithmic scaling are available by selecting the Scale pull-down

menu options.

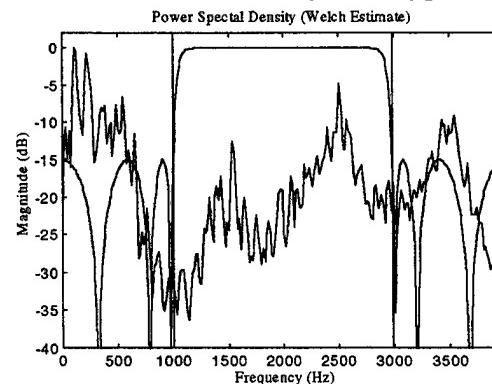
In filter design, cursor 1 (---) marks the lower cutoff frequency and cursor 2 (-.-) marks the upper cutoff frequency. For this exercise, we design a bandpass filter by selecting the Filter-Bandpass menu. After the filter transfer function is redrawn, you can see that the upper cutoff frequency is located at $fs/2$ as shown below.



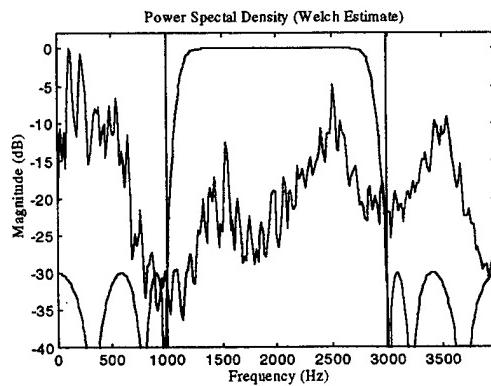
The upper cutoff frequency can be set by grabbing cursor 2 and dragging it to the desired upper cutoff frequency.



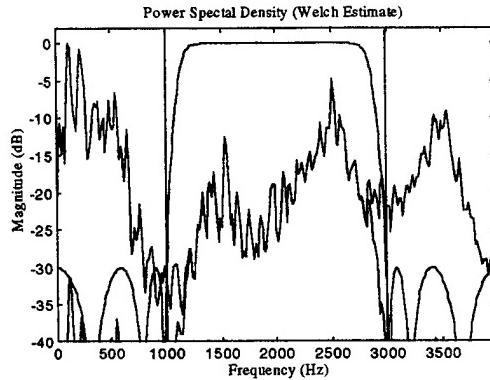
Now use the Filter menu to select a Chebychev Type II filter.



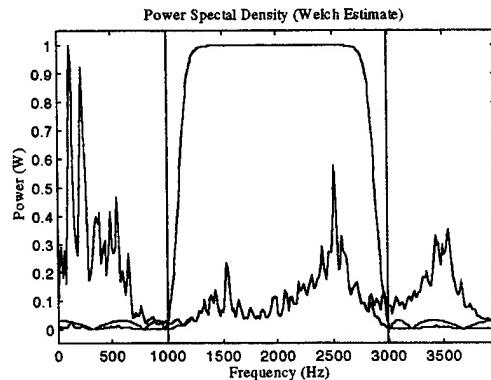
The default attenuation in the stopband for a Chebychev Type II filter is 15 dB. Using the stopband attenuation popupmenu, change the stopband attenuation to 30 dB.



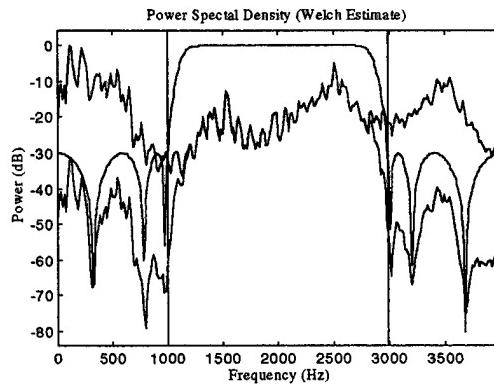
The filter design is now complete. Select the **Apply-Filter** menu to filter the signal. After the signal has been filtered, its spectrum is recomputed and redisplayed along with the filter transfer function. In the result shown below, the filter appears to have performed quite well.



Switch to a linear scale using the **Scale-Linear** menu.

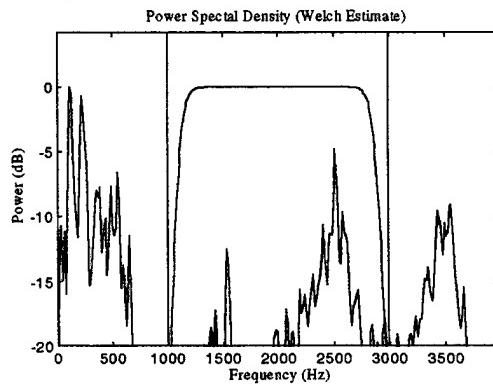


Now switch back to the logarithmic scale by using the **Scale-Logarithmic** menu.



Note the difference in the display. What has occurred is that the limits were reset such that the entire magnitude of the spectrum is visible. This is the same result as that obtained by selecting the **Scale-Full Magnitude** menu item. Note that after the signal is filtered, the magnitude of the frequencies corresponding to the nulls in the transfer function become very small and, as such, require a larger logarithmic axis to display. Sometimes nulls will extend so far down as to make it difficult to see details of the spectrum around 0 dB. Fortunately, the magnitude axis can easily be changed to effectively zoom in on 0 dB. Unlike zooming in on the time or frequency (horizontal) axis, the magnitude (vertical) axis can be zoomed in by simply clicking the mouse with the cursor somewhere inside the axis. On a logarithmic scale, clicking the mouse with the cursor below zero resets the lower axis limit to the magnitude the cursor was at when the mouse was clicked. The same operation with the cursor above 0 dB sets the upper axis limit. On a linear scale, any mouse click with the cursor inside the plot axis resets the upper axis limit. Note that the cursor must be in a blank portion of the screen within the axis when using this method (not near any curves or other lines).

The following is the result of zooming in on 0 dB using the above method (clicking the mouse around -20 dB).



See **sigfilt** for further discussion on the use of the **sigfilt** tool.

TUTORIAL

BPSK Modulation/Demodulation

This tutorial focuses on using commands from the SPC Toolbox to generate a BPSK signal containing a known message, simulate transmission of that signal through a noisy channel and then demodulate and recover the transmitted message from the “received” signal. While this tutorial uses BPSK as the modulation method, BFSK or OOK modulation could easily be substituted in its place.

This tutorial will use the following parameters:

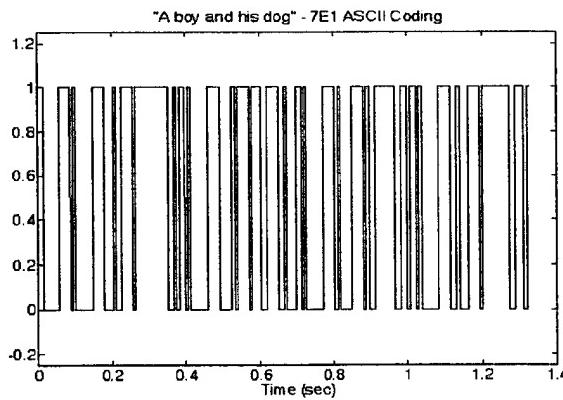
- BPSK modulation,
- 7-bit, even parity ASCII with one stop bit coding,
- 128 bits-per-second bit rate,
- Sampling frequency of 8192 Hz,
- Carrier frequency of 1024 Hz,
- 15 dB in-band, signal-to-noise ratio.

One popular way to transmit binary data is through the use of modems such as those found on most personal computers. A modem transmits data one character at a time. Each character is transmitted using one bit to mark the start of each new character and one or two bits to mark its end. The character itself is transmitted as either seven or eight bits with an optional parity bit as a means of performing some error checking. The function `str2masc` converts a character string to a vector of 1's or 0's representing the binary stream that would be sent if the string was transmitted by a modem.

```
msg = 'A boy and his dog';
m = str2masc(msg,7,'e',1);
m(1:10)
ans =
1 (start bit)
1 (data bits)
0
0
0
0
0
1
1 (parity bit)
1 (stop bit)
```

The `unipolar` command can be used to generate a baseband rendition of this signal.

```
fs = 8192; Rb = 128;
msgwave = unipolar(Rb,fs,m);
plot((0:length(msgwave)-1)/fs,msgwave);
set(gca,'YLim',[-0.25 1.25]);
title(' "A boy and his dog" - 7E1 ASCII Coding');
xlabel('Time (sec)');
```



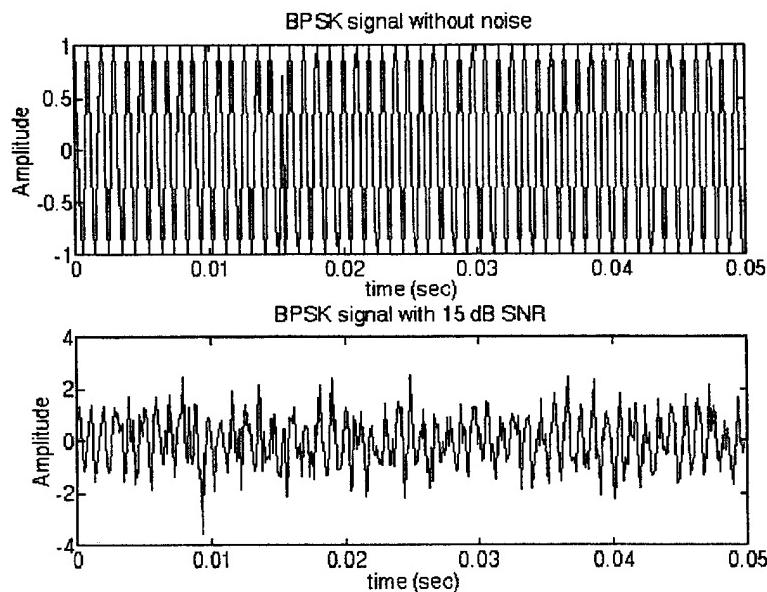
Two commands are available to directly create BPSK signals. The first is the `bpsk` command which creates random messages only. The second command, `bpskmsg`, is used to create a BPSK signal containing a specific binary message.

```
fc = 1024;
p = bpskmsg(Rb,fc,fs,m);
```

To simulate a noisy transmission channel, the signal is mixed with additive Gaussian white noise such that the in-band signal-to-noise ratio is 15 decibels. The `setsnrbw` command performs this function.

```
SNR = 15;
r = setsnrbw(p,randn(length(p),1),SNR,fc,Rb,fs);
```

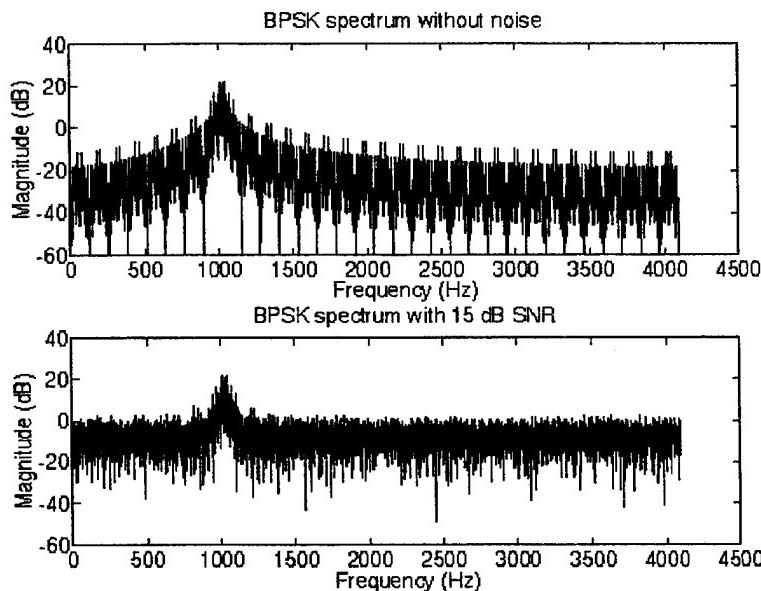
Taking a look at how this signal appears before and after “transmission” is easy to do with the `plottime` command.



```
subplot(2,1,1),plottime(p,0.05),
title('BPSK signal without noise');
xlabel('time (sec)'),ylabel('Amplitude');
subplot(2,1,2),plottime(r,0.05);
title(['BPSK signal with ' int2str(SNR) ' dB SNR']);
```

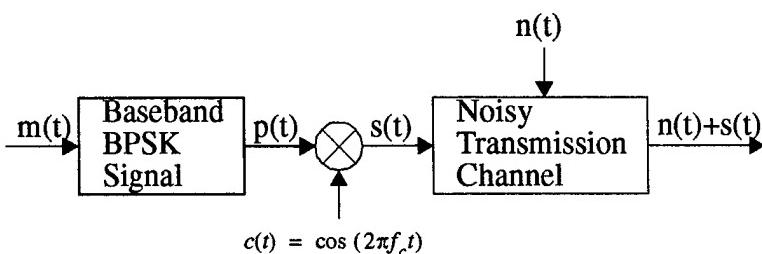
```
xlabel('time (sec)'); ylabel('Amplitude');
```

Also, plotting the noise-free and noisy power spectral densities (periodograms) is easy with the `lperigrm` command for a log scaled look (decibels) or `wperigrm` for a linear scaled look (relative watts).



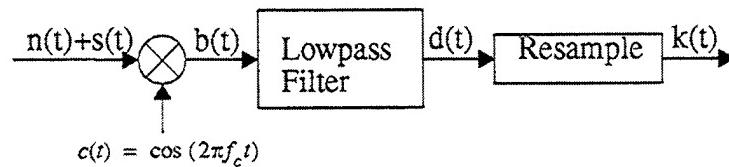
```
subplot(2,1,1),lperigrm(p)
title('BPSK spectrum without noise');
subplot(2,1,2),lperigrm(r)
title(['BPSK spectrum with ' int2str(SNR) ' dB SNR']);
```

Thus far, the following has been simulated:



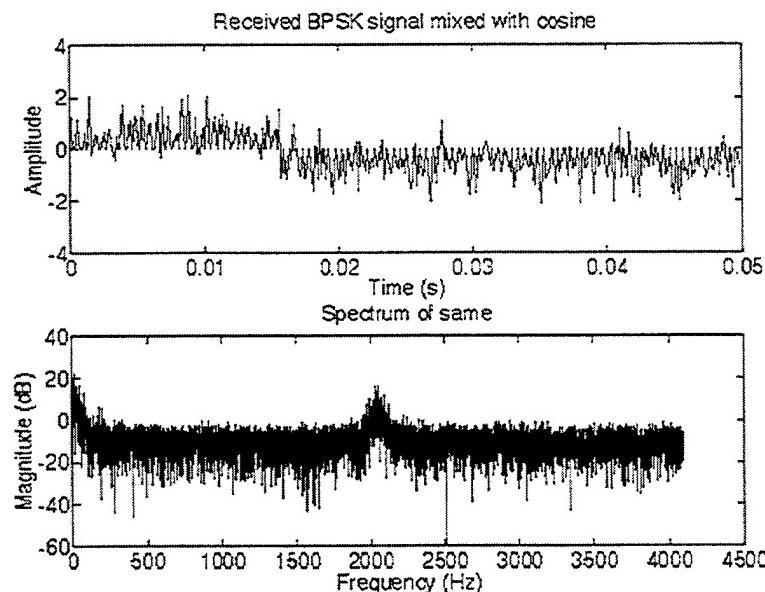
First, the message was encoded in binary using `str2masc` to produce $m(t)$. Second, $m(t)$ was keyed as a baseband BPSK signal, $p(t)$, and modulated onto a carrier, $c(t)$, creating the transmitted signal $s(t)$. The `bpskmsg` command actually did this operation by converting $m(t)$ into a polar signal using the `antipodal` command and then modulating this signal onto a sinusoidal carrier using double-sideband, suppressed-carrier modulation with the `dsbsc` command. Third, Gaussian white noise was added to the transmitted signal to simulate a noisy transmission channel using the `setsnrw` command.

The block diagram below outlines the system, known as a product detector, which is used to demodulate the “received” signal.



The reader may recall that demodulation of BPSK signaling can only be performed using coherent detection. To the uninitiated, this means that the sinusoidal signal shown in the diagram above must be in-phase with the carrier signal. This is easy to do as the `bpskmsg` command generates a carrier with zero phase shift. As long as the sinusoid above is created with zero phase shift (at the same sampling frequency), the sinusoid and the carrier are in-phase and hence, coherent detection can be performed.

```
t = 0:1/fs:(length(r)-1)/fs; c = cos(2 * pi * fc .* t');
b = c .* r;
```



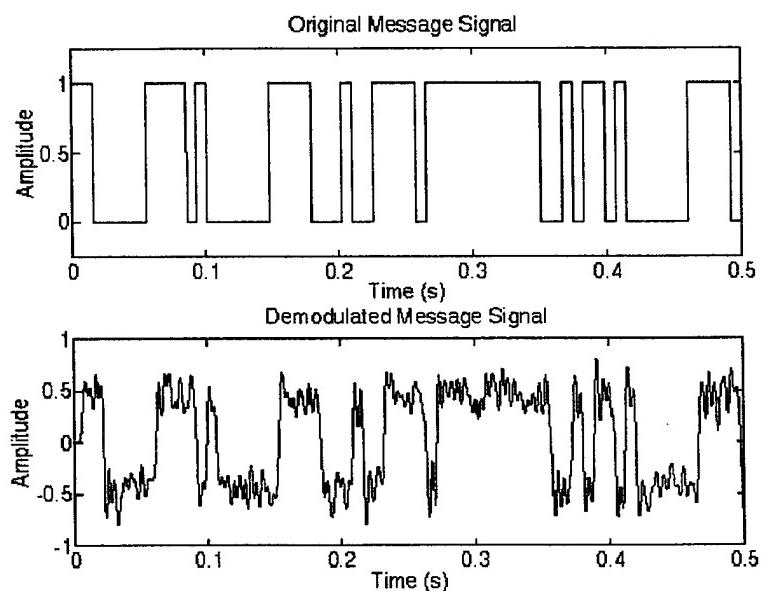
```
subplot(2,1,1),plottime(b,0.05);
title('Received BPSK signal mixed with cosine')
subplot(2,1,2),lperigrm(b)
title('Spectrum of same')
```

The above plot shows that the spectrum of $b(t)$ has an image at baseband and at twice the carrier frequency. Also note that the magnitude of the baseband image is greater than the image at twice the carrier frequency.

The next step requires filtering $b(t)$ using a lowpass filter. This filtering removes the image at twice the carrier frequency leaving a baseband polar signal. A number of factors come into play when selecting the type of filter and its cutoff frequency for use in this step. These include the roll-off characteristics of the filter and the number of signal

sidelobes to be included in the passband of the filter. Including more signal sidelobes will include more harmonic frequencies from the baseband signal (which is good) but it will also include more noise (which is bad). More harmonics frequencies would make a “squarer” waveform but more noise could offset that benefit. Obviously, trade-offs have to be made. Here, a tenth order, type 1 Chebychev lowpass filter with 1 dB of ripple in the passband is used. The passband will include the main lobe and one side-lobe.

```
[bb,aa] = cheby1(10,1,4*Rb/fs);
d = filter(bb,aa,b);
```



```
subplot(2,1,1),plottime(msgwave,0.5);
title('Original Message Signal');
subplot(2,1,2),plottime(d,.5);
title('Demodulated Message Signal');
```

As can be seen, the result indeed resembles a noisy polar waveform. Comparing this plot to the first plot of the binary waveform reveals a phase delay. This delay is caused by the non-causal property of the lowpass IIR filter used in this tutorial.

The final step is to recover the information. To do this, the signal $d(t)$ is resampled taking one sample per bit. To properly resample $d(t)$, care must be taken to sample each bit only when it has stabilized after a bit transition but before a transition to the next bit starts. Because of the filter delay, $d(t)$, can simply be sampled at times when bit transitions would have occurred (every f_s/R_b samples).

```
k = d(fs/Rb:fs/Rb:length(d)); % fs/Rb = samples-per-bit
```

Once resampled, $k(t)$ still represents an analog “voltage.” An analog system would use a threshold detector at this point to convert the analog signal to a binary signal. A logical comparison operator in Matlab can be used to simulate a threshold detector. In this case, the optimal threshold voltage is zero and everytime the signal $k(t)$ is greater than this value, it is considered a binary 1.

```
k = k > 0; % convert to binary
```

This results in a binary representation of the received message. This binary representation can be compared to the binary representation of the original message to compute the number of bit errors.

```
biterror = sum(k ~= m)
biterror =
0
```

Last, the `masc2str` command can be used to convert the received binary signal into a readable form.

```
masc2str(k,7,'e',1)
ans =
A boy and his dog
```

For further study:

1. Change various parameters in the above simulation and observe the results. Candidate parameters include:
 - Signal-to-noise ratio.
 - Modulation method.
 - Lowpass filter type, cutoff frequency, attenuation, ripple, etc.
2. Add interfering signals (i.e. voice, BFSK, etc.) in addition to the Gaussian white noise and attempt to recover the modulated information. This environment can be generated by first creating an environment containing the primary signal with Gaussian white noise (as in this tutorial), and then generating a second environment by creating the interfering signal and using the first environment as the noise input to the `setsnrw` command.
3. Pad the beginning and end of the signal with arbitrary amounts of white noise (scaled appropriately with the σ scaling factor returned by the `setsnrw` command). Write a command to “synch-up” with the signal in order to use coherent demodulation.
4. Experiment using error correcting codes and low SNR's.
5. Instead of sub-sampling the demodulated signal, try using an integrator or some other bit detection scheme.

Tool Reference

This section contains detailed descriptions of all interactive tools in the Signal Processing and Communications Toolbox. It begins with a list of functions grouped by subject area and continues with the reference entries in alphabetical order. Information is also available through the online help facility.

Interactive Signal Processing	
<code>sigedit</code>	Cut and past signal (vector) editing
<code>voicedit</code>	Enhanced speech signal editing
<code>gfilterd</code>	Graphical filter (pole/zero) design
<code>sigfilt</code>	FIR/IIR filtering for signals
<code>spect2d</code>	Two-dimensional spectral estimation
<code>spect3d</code>	Three-dimensional spectral estimation
<code>sigmodel</code>	Signal modeling tool.
<code>sspipam</code>	SSPI PAM program interface (signal generation) See help <code>sspipam</code>
<code>sspisxaf</code>	SSPI SXAF program interface (cyclostationary spectrum) See help <code>sspisxaf</code>

Interactive Graphical Tools	
<code>zoomtool</code>	Vector plot zoom/measurement
<code>zoomplay</code>	Vector sound playing
<code>zoomedit</code>	Vector editing
<code>p3dtool</code>	Three dimensional plot/zoom tool
<code>v3dtool</code>	Three-dimensional plot viewing/enhancement tool

aledsgn

Purpose

Adaptive Line Enhancer (ALE) Design Tool

Synopsis

`aledsgn(x)`

`aledsgn(x,fs)`

`[h] = aledsgn(parent,x,fs,'callback')`

Description

`aledsgn(signal,fs)` opens a Adaptive Line Enhancer Design Tool to design an ALE FIR filter for the signal x sampled at sampling frequency of fs Hz. If not supplied, the default sampling frequency is $fs = 1$.

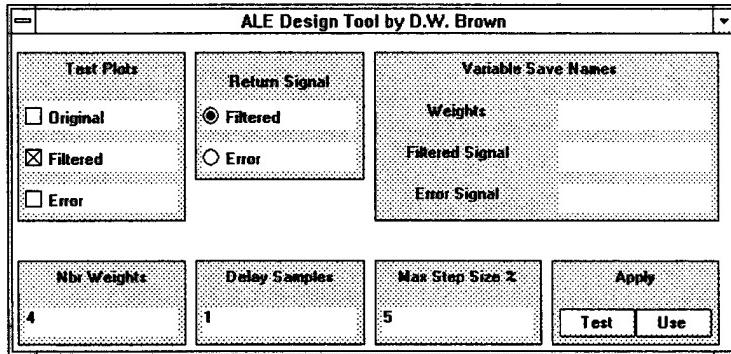
`[h] = aledsgn(parent,signal,fs,'callback')` returns the handle to the figure window the tool was opened in. This allows the ALE Design Tool to be called from other programs. *parent* is the handle to the figure window containing the calling application. The user has the capability to specify the signal to be made available to the callback function (as variable 'SPC_COMMON'). The sampling frequency argument, *fs*, is used only in labeling the time axis scale on test plots.

ALE Design Tool

The ALE Design Tool provides a graphical front end to the `lmsale` function. It allows the user to interactively change the number of weights, number of delay samples, and the percentage of the maximum step size used in each adaptation step (see the `lmsale` function for definition of the maximum step size). This is quite useful as these parameters quite often are found on a "trial and error" basis. Users can specify which test plots (original, filtered, and error signals) are displayed to study how effective the filter is for the given set of parameters. User can also specify variables names to store the filter weights, filtered signal and error signal to the MATLAB workspace making these variables available from the command prompt. The Test push button is used to filter the signal with the current parameters and view and/or save the results. The Use push button performs the same function only closing the ALE Design Tool. The test plot figure will remain displayed after closing the ALE Design Tool when the tool is called from the command prompt.

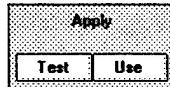
ALE Design Tool Screen

The ALE Design Tool screen consists of a several control groups containing radio buttons, check boxes, edit boxes and push buttons. These controls are used to select the test plots to display, select the signal to return to the calling program (if enabled), and enter the ALE filter parameters. Test plots are rendered in new windows after selecting the Test push button.



ALE Design Tool Specific Controls

Apply Test and Use Push Buttons



Purpose

Filter the signal using the current ALE filter parameters.

Use

1. Place the mouse pointer over the Test or Use push button and press the left mouse button.

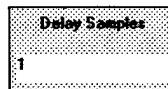
Result

The signal is filtered based on the current settings in the ALE Design Tool. If variables names have been entered into the Variable Save Names group, the appropriate variables are saved to the MATLAB workspace. If the Test push button was pressed and any Test Plots check boxes are checked, separate figure windows are opened containing each Test Plots signal. Test signals are displayed using zoomtool to allow the user to examine these signals in detail. If the Use push button was pressed, the signal is filtered using the current parameters and the tool window is closed.

Notes

- When the ALE Design Tool is called directly from the command prompt and the Use push button is pressed without any variable names entered or test plots selected, the tool is simply closed without filtering the signal.

Delay Samples Edit Box



Purpose

Sets the *decorrelation parameter* of the ALE. This parameter is the number of sam-

ples, d , used to remove the correlation between the noise component in the signal $x(n)$ and the delayed predictor input $x(n - d)$.

Use

1. Activate the edit box by clicking inside of the edit box with the mouse pointer.
2. Enter the desired decorrelation parameter and press return.

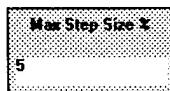
Result

The parameter is used the next time an **Apply** push button is selected.

Notes

- Valid variable range is one to the length of the signal minus one.

Max Step Size Percent Edit Box



Purpose

Set the percentage of the maximum step size used in each adaptation step (see the `lmsalg` function for definition of the maximum step size).

Use

1. Activate the edit box by clicking inside of the edit box with the mouse pointer.
2. Enter the desired percentage of maximum step size and press return.

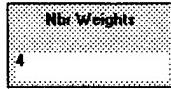
Result

The parameter is used the next time an **Apply** push button is selected.

Notes

- Valid variable range is $0 < \text{percentage} \leq 100$.

Number of Weights Edit Box



Purpose

Set the number of filter tap-weights (order).

Use

1. Activate the edit box by clicking inside of the edit box with the mouse pointer.
2. Enter the desired number of filter weights and press return.

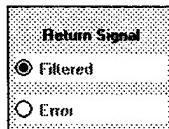
Result

The parameter is used the next time an **Apply** push button is selected.

Notes

- Valid variable range is one to the length of the signal minus one.

Return Signal Radio Button Group



Purpose

Specify the signal to be made available to the callback function (as the variable 'SPC_COMMON') when the ALE Design Tool is called from another program (such as sigedit). Note the filtered output of the ALE algorithm is the narrowband component of the input signal and the error signal is the wideband component of the input signal ("narrowband" and "wideband" being defined by the *correlation parameter*).

Use

1. Place the cursor on the desired signal (Filtered or Error) and click the left mouse button once to select.

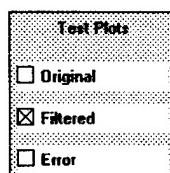
Result

Upon selecting the Apply Use push button, the global variable SPC_COMMON is set to equal the selected signal.

Notes

- This group is only available when the ALE Design Tool is called from another application program.

Test Plots Check Box Group



Purpose

Select the signals to be displayed (in separate windows) when the Apply Test push button is pressed.

Use

1. Place the cursor on the desired check box and click the left mouse button to turn the test plot on or off.

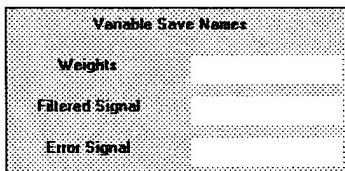
Result

The selected signals are displayed the next time the Apply Test push button is pressed.

Notes

- Test signal plots are displayed using zoomtool to allow the user to examine the signal in detail.

Variable Save Names Edit Box Group



Purpose

Enter the variable names to save the filter Weights, the Filtered Signal, and the Error Signal to the MATLAB workspace.

Use

1. Activate the edit box adjacent to the variable to save by clicking inside of the edit box with the mouse pointer.
2. Enter the desired variable name and press return.

Result

The filter weights, filtered signal and error signal (as appropriate) are saved to the MATLAB workspace the next time the Apply Test or Use push buttons are pressed.

Notes

- Entering a variable name for Weights and pressing return will propagate the same variable name into the Filtered Signal and Error Signal edit boxes. This avoids having to type the same name three times. To use different names for Filtered Signal and Error Signal, enter the Weights name first and then change the Filtered Signal and Error Signal names.
- When saved, the variable names are appended as follows:
 - "_w" for the filter weights.
 - "_y" for the filtered signal.
 - "_e" for the error signal.

Examples

Create a 60 Hertz tone in noise and then use the ALE Design Tool to design an adaptive line enhancer filter to recover the "narrowband" tone.

```
t = (0:1000) / 1000;
s = sin(2 * pi * 60 .* t);
sn = s + randn(1,length(t));
aledsgn(sn,1000);
```

Limitations

Dependent upon the input signal and the ALE parameters used, the filter may become unstable in which case the output signals will "blow up." In this case, the user will have to change the various ALE parameters until a stable filter is designed. This is usually a "trial and error" process. Decreasing the step size or increasing the number of

aledsgn

weights will usually have the most stabilizing effects.

See Also

`lmsale`, `zoomtool`, `sigedit`, `voicedit`

armadsgn

Purpose

AR/MA/ARMA model design tool.

Synopsis

```
armadsgn(x);
h = armadsgn(h,'PropertyName','PropertyValue',...)
```

Description

`armadsgn(x)` opens a AR/MA/ARMA Model Design Tool to model the signal `x`.

`h = armadsgn(h,'PropertyName','PropertyValue',...)` returns the handle to the figure window it was opened in. This is useful for programmers to keep track of `armadsgn` when it is called by a parent application. Such an example occurs when `armadsgn` is called by `sigmodel` or `spect2d`. Input argument `h` is the handle to the calling (parent) figure. Different properties are available to control the visibility of certain controls in the design tool. Visibility of these controls are dependent upon the use of `armadsgn` by the parent application and are determined by the programmer.

Note: No output arguments are supported. Model coefficients and the modeled signal and error signal must be saved by entering a name into the appropriate edit box in the Variable Save Names group.

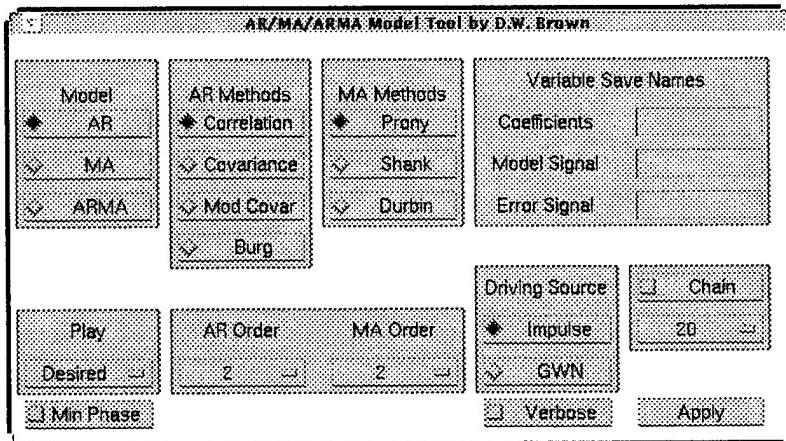
AR/MA/ARMA Design Tool

The AR/MA/ARMA Design Tool provides an interactive front-end to model Auto-Regressive (AR) and Auto-Regressive/Moving Average (ARMA) signals with the SPC Toolbox AR/MA/ARMA functions. With the AR/MA/ARMA Design Tool, you can:

- Model AR processes using the autocorrelation, covariance, modified covariance or Burg methods.
- Model MA processes using the Prony, Durbin or Shank methods.
- Model ARMA processes using any combination of the above AR and MA methods.
- Force the model polynomial to a minimum-phase condition.
- Drive the model with an impulse function or Gaussian white noise.
- Create the model and compute the error signal.
- Generate a chain of models. This feature is useful in modeling parts of speech.

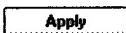
AR/MA/ARMA Design Tool Screen

The AR/MA/ARMA Design Tool screen consists of several control groups containing radio buttons, popup menus, check boxes, edit boxes and push buttons. These controls are used to select the modeling method and various parameters. Dependent upon the setting of the `CloseOnApply` property, the AR/MA/ARMA Design Tool screen may or may not close after the `Apply` push button has been pressed and the model created. The default mode is to close the tool after the `Apply` push button has been pressed.



AR/MA/ARMA Design Tool Specific Controls

Apply Push Button



Purpose

Generate the model.

Use

1. Place the mouse pointer over the **Apply** push button and press the left mouse button.

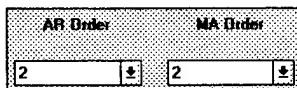
Result

The model is generated based on the current settings in the AR/MA/ARMA Design Tool. If variables names have been entered into the Variable Save Names group the appropriate variables are saved to the MATLAB workspace.

Notes

- The setting of the **CloseOnApply** property determines whether or not the tool is automatically closed after the model is generated. The default is to close the tool when the tool is started from the command prompt.

AR and MA Order Popup Menus



Purpose

Sets the order of the AR model (number-of-poles) and MA models (number-of-zeros).

Use

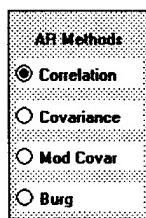
1. Place the cursor on the order popup menu and click the left mouse button once to open the menu.

2. Place the cursor on the desired order (1,2,3,4,6,8,10,12,14) and click the left mouse button. If the desired order is not available on the menu, the desired order can be entered by selecting the User menu item. After User is selected, an edit box appears in place of the popup menu. Enter the desired order into this edit box and press return.

Result

The AR or MA order is set.

AR Methods Radio Button Group



Purpose

Select the AR modeling method.

Use

1. Place the cursor on the desired modeling method (Correlation, Covariance, Modified Covariance, or Burg) and click the left mouse button once to select.

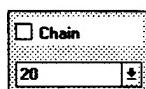
Result

The radio button of the desired modeling method is selected and all others are turned off. The desired model will be generated after the Apply push button is pressed.

Notes

- This setting is only valid when an AR or ARMA model has been selected (see Model Radio Button Group).

Chain Check Box



Purpose

Turn generation of a model signal (by filtering an impulse train or Gaussian white noise) on or off.

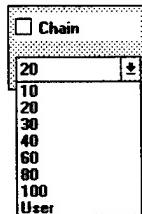
Use

1. Place the cursor on the Chain check box and click the left mouse button to turn the chain on or off.

Result

If checked, a model signal is generated using an impulse train equal to the number specified by the Chain Length popup menu (or an equivalent amount of Gaussian white noise).

Chain Length Popup Menu



Purpose

Set the number of impulses in the impulse train (or number of segments of Gaussian white noise) the model signal is generated with.

Use

1. Place the cursor on the Chain Length popup menu and click the left mouse button once to open the menu.
2. Place the cursor on the desired chain length (10, 20, 30, 40, 60, 80, or 100) and click the left mouse button. If the desired chain length is not available on the menu, the desired chain length can be entered by selecting the User menu item. After User is selected, an edit box appears in place of the popup menu. Enter the desired chain length into this edit box and press return.

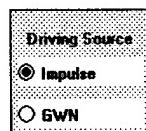
Result

The desired chain length is set.

Notes

- This feature is useful for modeling voiced phonemes in speech.

Driving Source Radio Button Group



Purpose

Choose between an impulse function or Gaussian white noise to drive the model generated.

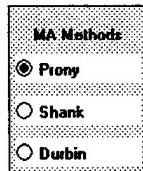
Use

1. Place the cursor over the desired driving source and click the left mouse button to select.

Result

The desired source is used when the model is applied.

MA Methods Radio Button Group



Purpose

Select the MA modeling method.

Use

1. Place the cursor on the desired modeling method (Prony, Shank, or Durbin) and click the left mouse button once to select.

Result

The radio button of the desired modeling method is selected and all others are turned off. The desired model will be generated after the **Apply** push button is pressed.

Notes

- This setting is only valid when an MA or ARMA model has been selected (see **Model Radio Button Group**).

Min Phase Check Box

Purpose

Turn forcing the MA model to minimum phase on or off.

Use

1. Place the cursor on the **Min Phase** check box and click the left mouse button until the check box is turned on or off as desired.

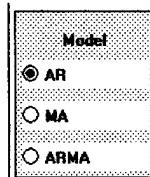
Result

The MA model transfer function polynomial is forced to a minimum phase condition after the model is generated.

Notes

- Forcing a minimum-phase condition will decrease the accuracy of the model if the process being modeled is not minimum-phase to begin with.

Model Radio Button Group



Purpose

Select between an AR, MA, or ARMA model.

Use

1. Place the cursor on the desired model (AR, MA, or ARMA) and click the left mouse button once to select.

Result

The radio button of the desired modeling method is selected and all others are turned off. The desired model will be generated after the **Apply** push button is pressed.

Play Popup Menu

Purpose

Play the Desired, Model, or Error signal over the workstations audio output.

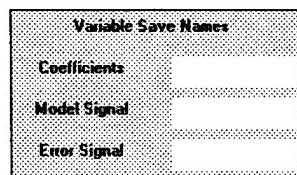
Use

1. Place the cursor on the Play popup menu and click the left mouse button once to open the menu.
2. Place the cursor on the menu item of the signal to play (Desired, Model, or Error) and click the left mouse button.

Result

- The selected signal is played over the workstation audio output. This feature is useful for comparison of the speech models to the original signal.

Variable Save Names Group



Purpose

Enter the variable name to save the Model Coefficients, the Model Signal, and the Error Signal to the MATLAB workspace.

Use

1. Activate the edit box adjacent to the variable to save by clicking inside of the edit

- box with the mouse pointer.
2. Enter the desired variable name and press return.

Result

Variables with names will be saved to the MATLAB workspace when the model is generated by selecting the **Apply** push button.

Notes

- The variable name for the coefficients is appended with an “_a” and “_b” for the AR and MA polynomials respectively.
- When the Burg AR method is used, the gamma coefficients are saved to the coefficients variable name appended with an “_g”.
- The variable names for the model and error signals are appended with an “_m” and “_e” respectively.
- Entering a variable name for Coefficients and pressing return will propagate the same variable name into the Model Signal and Error Signal edit boxes. This avoids having to type the same name three times. To use different names for Model Signal and Error Signal, enter the Coefficients name first and then change the Model Signal and Error Signal names.
- The underscore naming conventions discussed above allow the same variable name to be used for all saved variables. This makes it easy to distinguish variables belonging to the same model when multiple models are generated.
- To prevent a variable from being saved, clear the contents of the associated edit box.
- The default variable name when the AR/MA/ARMA Design Tool is called in a stand-alone mode (directly from the command-line) is temp.

Verbose Check Box

Purpose

Print detail model information in the MATLAB command window.

Use

1. Place the cursor on the Verbose check box and click the left mouse button until the check box is turned on or off as desired.

Result

The model method name and the model pole and zero polynomials, along with their roots in polar and rectangular coordinates, are printed in the MATLAB command window if the Verbose check box is checked when the **Apply** push button is pressed.

Notes

- The default setting when called from the command-line is off.
- Use the **diary** command to capture verbose output.

AR/MA/ARMA Design Tool Properties

The following properties can be set when **armadsgn** is started. Unlike MATLAB Handle Graphics Objects, the use of the **set** and **get** commands are not supported for these properties. The full property name must be used in specifying properties.

armadsgn

CallBack string

Control Action. This property specifies any legal MATLAB expression, including the name of any M-file or function to be called after the Apply push button has been selected and the model generated. This property is useful to the programmer when `armadsgn` is called as part of a larger GUI application (see `sigmodel` or `spect2d`).

Chain on | off

Display “Chain” control group. Controls display of the Chain control group. This group should be turned off when armadsgn is used for spectral analysis as in spect2d.

CloseOnApply yes | no

Selects whether or not to close the AR/MA/ARMA Design Tool after the **Apply** push button has been selected and the model generated. The default when **armadsgn** called from other than the command prompt is no.

Data vector

Signal to model. Sets the signal to be modeled equal to vector.

DrivingSource on | off

Display “Driving Source” control group. Controls display of the Driving Source control group. This group should be turned off when `armadsgn` is used for spectral analysis as in `spect2d`.

Minimum Phase on I off

Display “Minimum Phase” check box. Controls display of the Minimum Phase check box.

Verbose on | off

Set “Verbose” check box default value. Determines whether the Verbose check box to be checked (on) or unchecked (off) when the tool is opened. The default when armadsgn called from other than the command prompt is on.

Related Files

- **armagen.m** - Generates the model dependent upon the settings in the **armadsgn** tool.
 - **ar_METHOD.m** - SPC Toolbox AR/MA/ARMA modeling functions.

See Also

ar_corr, ar_covar, ar_mdcovar, ar_prony, ar_shank, ar_durbin, sigmodel, spect2d

gfilterd

Purpose

Graphical Filter Design Tool (polar form).

Synopsis

```
gfilterd;
gfilterd(b,a);
```

Description

`gfilterd` starts a Graphical Filter Design tool with an “allpass” transfer function (no poles or zeros).

`gfilterd(b,a)` starts a Graphical Filter Design tool with the predefined transfer function specified by the arguments *a* and *b*. The arguments *a* and *b* are of the same form as those returned by any of the MATLAB *Signal Processing Toolbox* filter design functions such as `butter` or `cheby1`. Once a pre-defined transfer function has been loaded, poles and zeros can be added, deleted, and/or moved about in order to “fine tune” or study the filter transfer function.

Note: No output arguments are supported. Any completed filter design must be saved using the **Save** push button described below.

Graphical Filter Design Tool

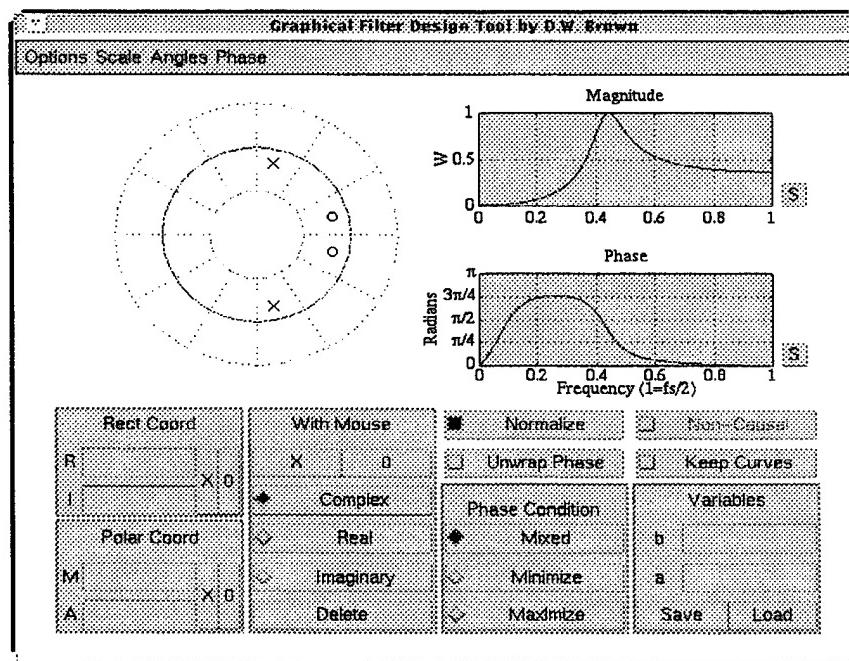
The Graphical Filter Design Tool provides an interactive, graphical environment within MATLAB to design digital filters through individual (complex-conjugate pair) placement of poles and zeros on a pole-zero plot of the filter transfer function. With the Graphical Filter Design Tool, you can:

- Place complex, real or purely imaginary poles and zeros onto the pole-zero plot using the mouse.
- Place complex, real or purely imaginary poles and zeros onto the pole-zero plot by specifying the real and imaginary values.
- Place complex, real or purely imaginary poles and zeros onto the pole-zero plot by specifying the magnitude and angle (degrees or radians) values.
- Move pole and zero locations using the mouse.
- Delete poles or zeros from the design using the mouse.
- Minimize and maximize the filter transfer function phase response.
- Force non-causal transfer functions causal.
- View the filter frequency and phase response as the design progresses.
- Generate the impulse response of the filter.
- Print the filter parameters.
- Save the filter transfer function coefficients to the base workspace.

Graphical Filter Design Tool Screen

The Graphical Filter Design Tool screen consists of a MATLAB figure window containing three axes and a number of controls (detailed below). The first axis contains a polar grid with radial lines every thirty degrees and circular lines at 0.5, the unit circle and at 1.5. The grid lines and unit circle colors can be customized by editing the `SPC_POLAR_AXIS` and `SPC_UNIT_CIRCLE` variables in the `spcolors.m` configuration

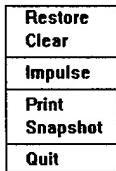
file. Plots containing the transfer function's magnitude and phase response curves are located to the right of the polar plot. The background color of these axis can be customized by editing the `SPC_GFILT_AXIS` variable in `spcolors.m`. The horizontal axis of both response curves correspond to frequency and is normalized such that the value one equates to the Nyquist frequency, $fs/2$. This is the same normalization as that used by the filter design functions contained in the MATLAB *Signal Processing Toolbox*. The vertical axis of the magnitude plot can be displayed in linear or decibel units via the Scale pull-down menu. The height of the magnitude curve can be normalized such that the highest peak is equal to one by checking the Normalize checkbox. The vertical axis of the phase plot corresponds to phase shift and can be expressed in units of degrees or radians through use of the Angles pull-down menu. It can range from -180 to +180 or from $-\pi$ to $+\pi$ and any values outside these limits will be "wrapped" to the top or bottom if the Unwrap check box is unchecked. Note that when the phase angle is displayed in degrees, the value specified in the angle argument of the polar coordinate entry group is in degrees. Likewise, when the phase angle axis is displayed in radians the angle argument is specified in radians. Please refer the limitations of the unwrap command in the MATLAB *Reference Guide*.



Graphical Filter Design Tool Menus

The Graphical Filter Design Tool contains the following pull-down menus.

Options Menu



Options-Restore Menu Item

Purpose

Restore the original filter coefficients specified as input arguments.

Use

1. Pull down the Options menu.
2. Select the Restore menu item.

Result

If gfilterd was called with input arguments, the transfer function specified by the calling arguments is restored. All design changes are lost if not previously saved using the Save push button. If no input arguments were used, the Restore push button has the same effect as the Clear menu item.

Options-Clear Menu Item

Purpose

Clear all poles and zeros from the filter design, effectively setting the transfer function to equal the constant 1.

Use

1. Pull down the Options menu.
2. Select the Clear menu item.

Result

All poles and zeros are cleared. All design changes are lost if not previously saved using the Save push button.

Options-Impulse Menu Item

Purpose

Generate a plot of the first 100 points of the current filter's impulse response.

Use

1. Pull down the Options menu.
2. Select the Impulse menu item.

Result

A graphics window which contains a plot of the impulse response is opened.

Notes

- The filter's stability can be determined from the impulse response.
- The graphics window containing the impulse response is forgotten by *gfilterd* after it is created. Thus, it can be treated as any MATLAB graphics window. The user has full control over this figure window, allowing changing the title, labels, scaling, printing, etc. Issue a `figure(N)` command (where N is the figure number) to make this the current figure.

Options-Print Menu Item

Purpose

Print the filter parameters in the MATLAB command window.

Use

1. Pull down the Options menu.
2. Select the Print menu item.

Result

Current pole and zero locations are printed in both rectangular and polar coordinates along with the transfer function polynomial coefficients in the MATLAB command window.

Notes

- The state of the Phase Condition radio buttons determines whether the parameters of a minimized, maximized or "mixed" phased system is printed.
- The filter parameters can be saved to a text file by using the `diary` command before printing or by cutting from the command window using the operating system window cut and paste facility, if available.
- The filter polynomials are printed using the same order polynomials for the numerator and denominator. Thus, the sample output below can be interpreted as either

$$H(z) = \frac{z^4 - 0.0705z^3 - 1.1987z^2 - 0.188z + 0.5517}{z^4 - 1.4097z^3 + 1.9014z^2 - 1.0632z + 0.5750}$$

or

$$H(z) = \frac{1-0.0705z^{-1} - 1.1987z^{-2} - 0.188z^{-3} + 0.5517z^{-4}}{1-1.4097z^{-1} + 1.9014z^{-2} - 1.0632z^{-3} + 0.5750z^{-4}}$$

Note the second form is that required for use in the `freqz` and `filter` functions.

```

-----
Filter Polynomials
-----
Zeros
  Real      Imag      Mag      Angle
-0.8018    0.2996    0.8559    0.8862
-0.8018   -0.2996    0.8559   -0.8862
  0.8370    0.2291    0.8678    0.0850
  0.8370   -0.2291    0.8678   -0.0850

Poles
  Real      Imag      Mag      Angle
  0.1850    0.8458    0.8658    0.4314
  0.1850   -0.8458    0.8658   -0.4314
  0.5198    0.7049    0.8758    0.2977
  0.5198   -0.7049    0.8758   -0.2977

Zero Polynomial
  1.0000   -0.0705   -1.1987   -0.0188    0.5517

Pole Polynomial
  1.0000   -1.4097    1.9014   -1.0632    0.5750

```

Options-SnapShot Menu Item

Purpose

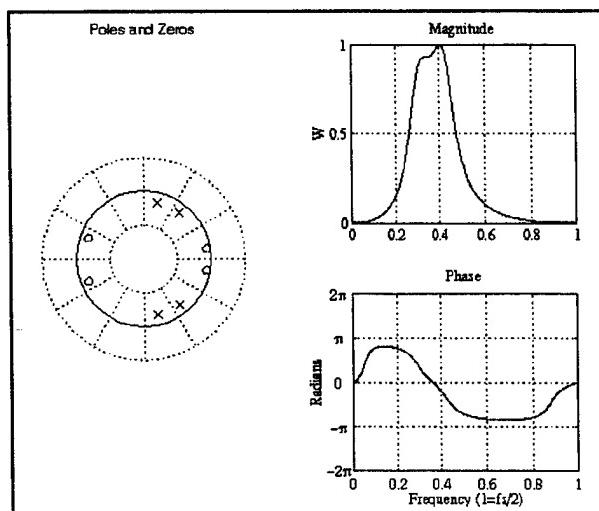
Recreate the transfer function pole-zero and frequency response plots in a new graphics window.

Use

1. Pull down the Options menu.
2. Select the Snapshot menu item.

Result

The current plots are recreated in a new graphics window. The user can then treat this window as any other MATLAB graphics window. Hardcopy can be obtained using the MATLAB print command. An example hardcopy is shown below.



Options-Close Menu Item

Purpose

Close the tool.

Use

1. Pull down the Options menu.
2. Select the Close menu item.

Result

The tool figure window is closed.

Notes

- The tool can also be closed using the windowing system (X, MS-Windows, Motif, etc) method of closing windows.

Scale Menu



Purpose

Select the scaling of the frequency response magnitude plot (logarithmic or linear).

Use

1. Pull-down the Scale menu and select the desired Linear or Logarithmic menu item.

Result

The transfer function response is redisplayed with the desired magnitude scaling set.

Angles Menu



Purpose

- Select the scaling of the frequency response phase plot (degrees or radians).
- Select the entry option for the Angle edit box in the Polar Coordinate Entry Group

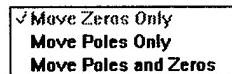
Use

1. Pull-down the Angle menu and select the desired Radians or Degrees menu item.

Result

The phase response is redisplayed and entry for the angle edit box is set to the selected option.

Phase Menu



Purpose

- Select whether poles, zeros or both are to be moved to their $1/z^*$ position when setting the phase condition.

Use

1. Pull-down the Phase menu and select one of the Move Zeros Only, Move Poles Only, or Move Poles and Zeros menu items as desired.

Result

The transfer function response is redisplayed with the phase conditioning.

Notes

- Minimizing or maximizing the poles and zeros phase conditions by moving them to their conjugate reciprocal locations does not ensure the system will be a minimum or maximum phase system. Refer to a linear systems text to review criteria for minimum and maximum phase systems.

Graphical Filter Design Tool Controls

Dragging Poles and Zeros

Purpose

Drag poles and zeros to new locations.

Use

1. Move mouse pointer over a pole or zero to be moved (or any one member of a complex-conjugate pair).
2. Press the left mouse button and hold. The pole or zero will double in size and the cursor pointer will turn into a cross hair indicating the item has been selected.
3. While holding the left mouse button down, drag the pole or zero to its new location.
4. When the pole or zero is in its new location, release the left mouse button.

Result

If the pole or zero is part of a complex-conjugate pair, the other member of the pair is moved to the conjugate location and the transfer function response is recomputed and displayed.

Notes

- Complex poles and zeros can be dragged to any location on the complex plane. Real and purely imaginary poles and zeros can only be dragged along their respective real or imaginary axis. An odd number of poles or zeros located at the origin will allow dragging one pole or zero along the real axis only. An even number will allow dragging a complex conjugate pair along the imaginary axis only. The desired object to

be moved (pole or zero) may not be “grabbed” depending on the object stacking order if both poles and zeros are located at the origin. When this situation occurs, the only recourse is to delete poles or zeros until the desired object can be grabbed and then reconstructing the system after moving the desired object. If virtual poles are present (see note below and notes under the Non-Causal push button section), the system may have to temporarily be made a non-causal system before objects located at the origin can be moved.

- Dependent upon the setting of the Phase menu, if a pole or zero is dragged outside the unit circle while the Minimize radio button is selected, the pole or zero will be shifted to the corresponding minimum phase location. The opposite applies to poles and zeros dragged inside the unit circle when the Maximize radio button is selected.
- When a non-causal system is forced to a causal state (see the Non-Causal push button), “virtual” poles are placed at the origin to bring the pole order up to the zero order. These poles cannot be dragged to new locations and attempts to do so will generate the error message “*gfilterd: Cannot move virtual poles.*” Virtual poles are removed from the system as additional poles are added to the system or when zeros are deleted by the user. Instead of attempting to move virtual poles, simply place new poles in the desired locations instead. The term “virtual” is used here simply to identify poles placed on the polar plot by the Graphical Filter Design tool and not the user.
- See the caution under Delete push button below.

Delete Push Button



Purpose

Delete poles and zeros using the mouse.

Use

1. Place the mouse cursor over the push button and press the left mouse button.
2. The mouse cursor will change into a crosshair and the x-axis label will change to “Click cursor on item to delete.”
3. Move the crosshairs over one of the poles or zeros in the complex-conjugate pair on a real pole or zero and select the left mouse button.

Result

The complex-conjugate pair or a single real pole or zero is deleted and the transfer function recomputed and displayed. The mouse cursor changes back into an arrow and the x-axis label is cleared.

Notes

- Real poles and zeros are deleted one-at-a-time. Complex conjugate poles and zeros are deleted as pairs. If multiple objects are located in the same location (i.e. the origin), only the top “layer” is deleted unless the object is located on the real axis or at the origin.
- When a non-causal system is forced to a causal state (see the Non-Causal push button), “virtual” poles are placed at the origin to bring the pole order up to the zero

order. These poles cannot be deleted and attempts to do so will generate the error message “*grafilterd: Cannot delete virtual poles.*” Virtual poles are removed from the system as additional poles are added to the system or when zeros are deleted by the user.

- In general, notes applying to dragging poles and zeros also apply to deleting them with the mouse.
- To cancel a delete operation, select the Delete push button a second time.

Caution

If a pole or zero is located directly on a polar grid line in the pole-zero plot, it may not be selectable if the mouse has been clicked with the pointer above the construction line. This is due to the fact that MATLAB “bumps” the construction line to the top of the stacking order, and since its selection area completely overlaps the pole or zero line object, the pole or zero is no longer accessible. To bring the pole or zero object on top of the construction line force the plot to be redrawn by re-selecting the current phase condition radio button. This operation will re-render the axes with the pole and zero line objects on top of the polar grid lines.

Keep Curves Check Box



Purpose

Keep previous magnitude and frequency response curves.

Use

1. Place the mouse cursor over the check box and check or uncheck the box with the left mouse button.

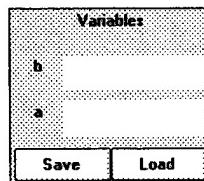
Result

Subsequent transfer function response curves are added to the magnitude and phase axes.

Notes

- Unchecking the Keep Curves check box deletes previously saved curves.
- As poles and zeros are added, the previous transfer function curves are retained as dotted lines and their colors cycle through the order specified in the SPC_COLOR_ORDER variable defined in the configuration file spcolors.m. The current transfer function color is specified by the SPC_TRANSFER variable.
- The Keep Curves check box is automatically unchecked whenever Options-Restore, Options-Clear, Scale-Linear, Scale-Logarithmic, Angle-Degrees, Angle-Radians menu items are selected, and whenever the Normalize check box or Load push button is selected.

Load Push Button



Purpose

Load the filter transfer function coefficients from the MATLAB workspace variables with the names specified in the "a" and "b" variable edit boxes.

Use

1. Enter the variable names in the "a" and "b" edit boxes.
2. Place the cursor on the Load Push Button and click the left mouse button.

Result

The transfer function is loaded into the graphical filter design tool and the transfer functions is recomputed and its response displayed.

Magnitude and Phase Plot "S"napshot Push Buttons



Purpose

Create a snapshot figure with zoomtool attached allowing measurement of transfer function curve features (i.e. peak frequency) and/or printing of the plot.

Use

1. Place the mouse cursor over the "S" push button nearest the desired plot axes and press the left mouse button.

Result

A new figure window is created containing the plot and zoomtool. zoomtool can be used to closely inspect the response curve. See zoomtool for information on its operation. The new figure window is forgotten by gfilterd allowing it to be treated as any other MATLAB figure window.

Non-Causal Check Box



Purpose

Force a system to have the same number of poles and zeros (polynomials of the same degree).

Use

1. Place the mouse cursor over the check box and check or uncheck the box with the

left mouse button.

Result

Poles are added so that the transfer function numerator and denominator polynomials are of the same order. “Virtual” poles are thus added and displayed at the origin.

Notes

- The Non-Causal check box is only enabled when the zeros outnumber the poles in the filter transfer function.
- “Virtual” poles are simply place holders to force the transfer function to be causal. They cannot be moved or deleted from the origin. For each pole added to or zero deleted from the transfer function, a “virtual” pole is removed until the transfer function is causal in its own right. At that time, the Non-Causal check box will be disabled.

Normalize Check Box



Purpose

Normalize the magnitude response curve.

Use

1. Place the mouse cursor over the check box and check or uncheck the box with the left mouse button.

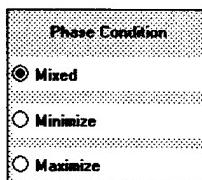
Result

The filter magnitude response curve is normalized by dividing through by its peak value and the filter response is recomputed and displayed.

Notes

- The Normalize check boxes the magnitude of the filter transfer function.

Phase Condition Radio Buttons



Purpose

Attempt to minimize or maximize the phase of the filter transfer function by moving the poles and/or zeros to their conjugate reciprocal locations.

Use

1. Check whether to move the poles, zeros or both from the Phase menu.
2. Place the mouse cursor on the desired phase condition action (Minimize, Maximize or

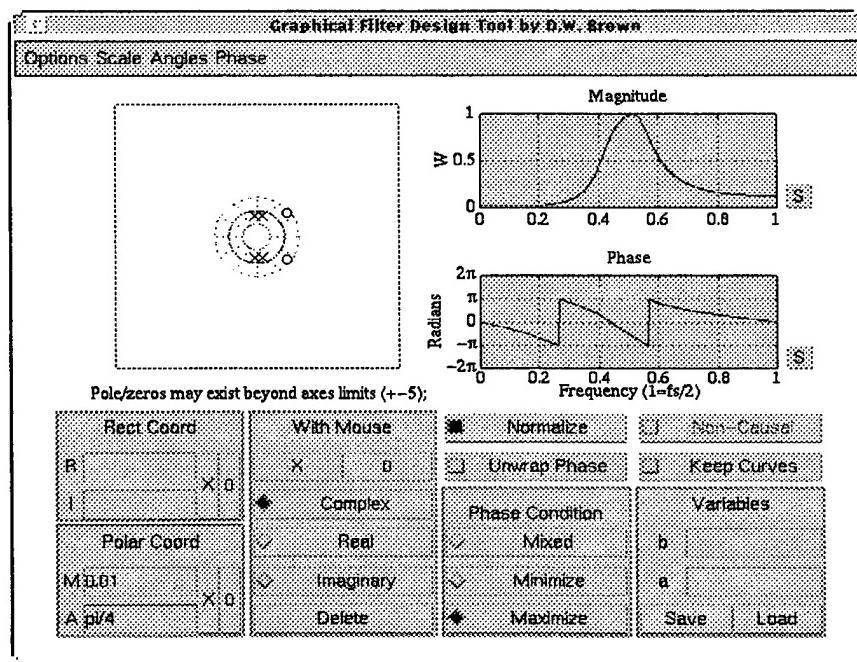
Mixed) and press the left mouse button.

Result

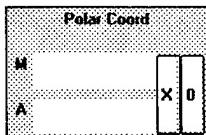
A new transfer function is computed and displayed according to the desired selection.

Notes

- In general, minimized phase systems have all their poles and zeros located inside the unit circle and maximized phase systems have all their poles and zeros located outside the unit circle. A “Mixed” phase systems does not attempt to change the phase condition from that naturally occurring due to the placement of the poles and zeros.
- Selecting Minimize or Maximize does not affect the internal storage of the pole or zero locations, therefore selecting Mixed after selecting Minimize or Maximize will return the poles and zeros to their original location (except those dragged to new locations while Minimize or Maximize is in effect).
- See the minphase and maxphase functions for the algorithm used in moving poles and zeros to their conjugate reciprocal locations.
- The state of the Phase Condition radio buttons affects the Print and Save menu items (see below).
- When poles and zero close to the origin are moved to their conjugate reciprocal locations, they can, of course, approach infinity. This correspondingly causes the polar axis to shrink. To prevent the polar axis from shrinking too far down to be useful, the variable SPC_GFILT_LIMIT in the *spcolors.m* configuration file can be set to the magnitude beyond which poles are zeros are not be shown on the polar axis. When this limit is exceeded by a pole or zero, a box will appear around the polar axis and the warning “*Poles/zeros may extend beyond axes (+5)*” where “5” is the value specified by SPC_GFILT_LIMIT.



Polar Coordinate Entry Group



Purpose

Add a pole or zero using polar coordinates (magnitude and angle).

Use

1. Enter the magnitude and angle of one pole or zero in the complex-conjugate pair.
2. Select the group's "X" push button to place a pole. Select the group's "O" push button to place a zero.

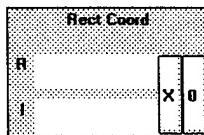
Result

The desired pole or zero is added to the pole-zero plot and the transfer function response is recomputed and displayed.

Notes

- Numeric expressions (i.e “pi/4”, “45/2”, “sqrt(2)”, etc.,) can be entered into the edit boxes.
- Angle entry is tied to the currently checked Degrees or Radians menu item on the Angles pull-down menu.
- Blank edit boxes default to zero.

Rectangular Coordinate Entry Group



Purpose

Add a pole or zero using rectangular coordinates (real and imaginary parts).

Use

1. Enter the real and imaginary parts of one pole or zero in the complex-conjugate pair. Enter only the real or only the imaginary part of a purely real or imaginary pole or zero.
2. Select the group's "X" push button to add a pole. Select the group's "O" push button to place a zero.

Result

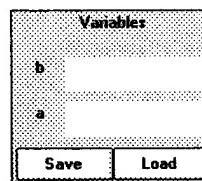
The desired pole or zero is added to the pole-zero plot and the transfer function response is recomputed and displayed.

Notes

- Numeric expressions (i.e “1/2”, “3*2/5”, etc.,) can be used in the edit boxes.

- Blank boxes default to zero.

Save Push Button



Purpose

Save the filter transfer function coefficients to the MATLAB workspace under the names specified in the "a" and "b" variable edit boxes.

Use

1. Enter the variable names in the "a" and "b" edit boxes.
2. Place the cursor on the Save Push Button and click the left mouse button.

Result

The transfer function is saved to the MATLAB workspace.

Notes

- Once saved to the MATLAB workspace, these variables can be used as inputs to the MATLAB *Signal Processing Toolbox* filter and freqz commands.
- Pole and zero locations can be recovered by first saving the transfer function coefficients using the Save menu and then using the MATLAB roots command to compute the pole and zero locations from the transfer function polynomials.
- The pole and zero polynomials are saved in the form required for input into the freqz and filter function (negative powers of Z).

Unwrap Phase Check Box



Purpose

Unwrap the phase angle curve.

Use

1. Place the mouse cursor over the check box and check or uncheck the box with the left mouse button.

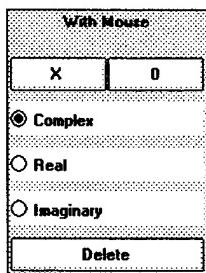
Result

The phase response curves are unwrapped when the Unwrap Phase check box is checked.

Notes

- The MATLAB unwrap command is used. See the MATLAB *Reference Guide* for limitations of the unwrap command.

“With Mouse” X and O Push Buttons



Purpose

Place a pole or zero on the pole-zero plot using the mouse cursor.

Use

1. Set the pole/zero property using the radio buttons describe below.
2. Select placement of a pole or a zero with the “X” or “O” push button. After selection, the cursor turns into a crosshair and the x-axis label of the pole-zero plot changes to “Mark pole location with cursor” or “Mark zero location with cursor” as appropriate.
3. Move the crosshair to the desired pole or zero location and press the left mouse button.

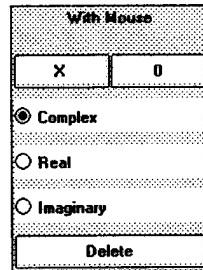
Result

The pole or zero is placed on the pole-zero plot and the transfer function response is recomputed and redisplayed. The mouse cursor changes back into an arrow and the x-axis label is cleared.

Notes

- To cancel placement of a pole or zero, click anywhere outside an imaginary rectangle surrounding the polar plot.
- Only the real or imaginary coordinate is of importance when setting purely real or imaginary poles or zeros.
- When a pole or zero is placed inside the unit circle with the Maximize radio button set, it will be automatically move to its conjugate reciprocal location in order to maximize the phase response. Selecting the Mixed or Minimize phase condition radio button will move the pole or zero to its placed location. The opposite applies to poles and zeros placed outside the unit circle when the Minimize radio button is active.
- Poles and zeros outside the currently displayed pole-zero axis limits must be entered using one of the coordinate entry groups described above.

“With Mouse” Property Radio Buttons



Purpose

Select whether poles and zeros placed on the pole-zero plot are complex, real or purely imaginary.

Use

1. Place the mouse cursor on the appropriate radio button and press the left mouse button.

Result

The radio button is selected and the chosen property is applied the next time a pole or zero is placed using the mouse.

Notes

- Complex poles and zeros are placed as complex-conjugate pairs.
- Real poles and zeros are placed as a single pole or zero on the real axis. In placing real poles and zeros, only the location of the cursor with respect to the real (horizontal) axis is important.
- Imaginary poles and zeros are placed as complex-conjugate pairs. In placing purely imaginary poles and zeros, only the location of the cursor with respect to the imaginary (vertical) axis is important.

Examples

Design a cheby2 bandpass filter and load the transfer function into gfilterd. Make the filter an eighth-order with 15 dB of attenuation in the stop band and have a passband from 0.4 to 0.7.

```
[b,a] = cheby2(4,15,[0.4 0.7]);
gfilterd(b,a);
```

Algorithm

The pole and zero locations are stored using rectangular coordinates. When the transfer function polynomials are formed, each pole or zero is placed into a vector of the form $[1 -p]$ or $[1 -z]$ and is then convolved with all other poles or zeros. Due to machine round-off errors, complex coefficients may occur. This is prevented in the final output by using only the real portion of the polynomial coefficients. A 512-point FFT is used to evaluate the frequency response curves.

See the `minphase` and `maxphase` functions for the algorithm used in creating minimum or maximum phase systems.

Related Files

- `gfilterd.m` - Graphical Filter Design Tool callback function library.

See Also

`minphase`, `maxphase`

sigedit

Purpose

Visual signal editing tool.

Synopsis

```
sigedit(x)
sigedit(x,fs)
sigedit('filename.ext')
sigedit('filename',bits)
h = sigedit(x,fs)
```

Description

`sigedit(x)` opens a signal editing tool containing the time-domain signal *x*.

`sigedit(x,fs)` sets the sampling frequency to *fs*. The default sampling frequency is 8192 Hz. A sampling frequency of 1 Hz will display the time axis in sample number versus time.

`sigedit('filename.ext')` - Opens the file specified by '*filename.ext*' and loads the signal stored within the file directly into the tool. File formats with the extensions **.au*, **.voc*, **.wav*, and **.tim* can be specified by the '*filename.ext*' option only (be sure to include the extension). Path names are required when the file is not in the current directory. The sampling frequency is set to that stored in the file for **.au*, **.voc* and **.wav* formats and to *fs* = 1 Hz for **.tim* files. See the function `readsig` for more information on the functions used to read signal files.

`sigedit('filename',bits)` - Opens the file specified by '*filename*' and loads the signal stored within directly into the tool as a flat integer formatted file. The number of bits-per-integer is specified by the *bits* argument where *bits* is either 8, 16, or 32. Integer formats must be stored as signed integers to be read correctly. Multi-byte integers (16 and 32) are assumed to be compatible with the format used by the workstation in use ("Little Endian" and "Big Endian" byte ordered files are not compatible). Path names are required when the file is not in the current directory. The sampling frequency is set according to the file type loaded. See the File-Load menu item description for more information.

`h = sigedit(x,fs)` returns the handle to the figure window it was opened in. This is useful for programmers to keep track of `sigedit` when it is called by a parent application. Such an example occurs when `sigedit` is launched from `voicedit`. See `zoomprog.m` for programming hints.

Note: No output arguments are supported. Edited signals must be saved using the Save menu item under the Workspace pull-down menu.

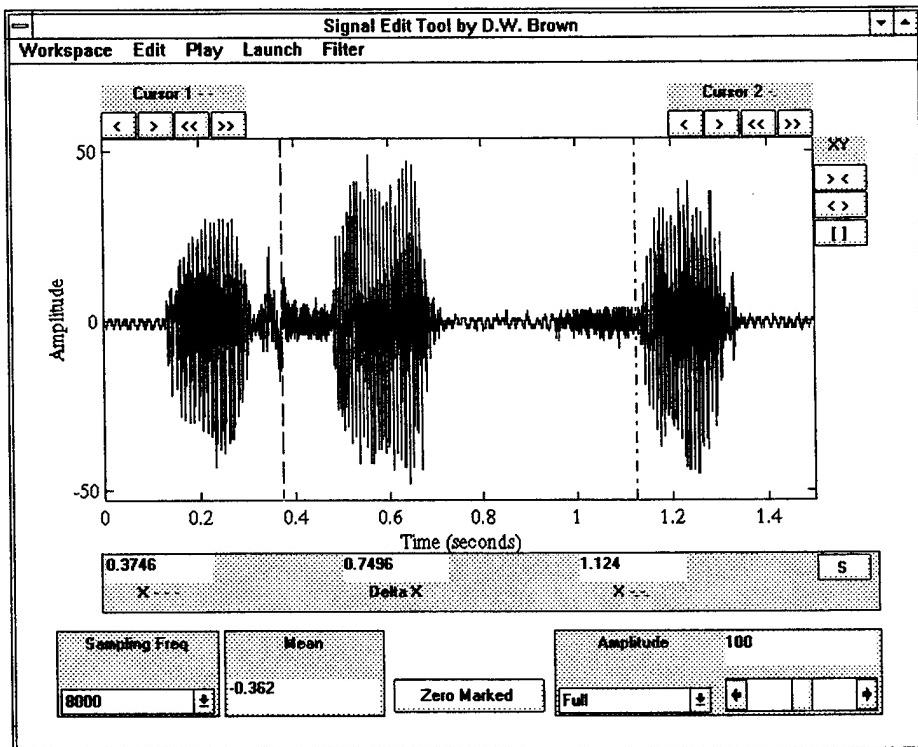
Signal Edit Tool

The Signal Edit Tool provides an interactive, visual environment within MATLAB to edit signals represented as row or column vectors. With the Signal Edit Tool, you can:

- Cut unwanted sections of a signal,
- Crop unwanted sections of a signal,
- Rearrange sections of a signal by cut-and-paste,
- Adjust the amplitude (volume) of all or a portion of a signal,
- Introduce periods of silence into a signal,
- Zoom on portions of a signal for a more detailed study.

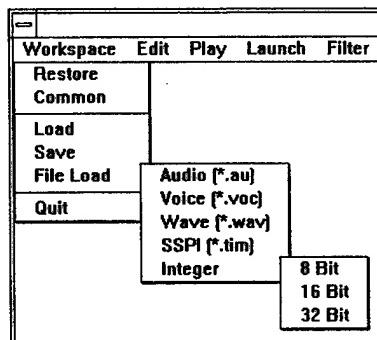
Signal Edit Tool Screen

The Signal Edit Tool screen consists of a set of pull-down menus and a group of controls along with an axis containing the signal to be edited. The axis uses zoomtool to provide zoom capabilities along with cursor controls and cursor position readouts. Only the vertical cursors of zoomtool are displayed. The user can use the scroll capability of zoomtool as an aid in finding a particular portion of the signal to edit after zooming in on the signal. See the zoomtool function description for more information on cursor movement and zoom/pan operations.



Signal Edit Tool Pull-Down Menus

Workspace Menu



Workspace-Restore Menu Item

Purpose

Restore the original signal.

Use

1. Select the Restore menu item from the Workspace pull-down menu.

Results

The original signal is restored.

Notes

- Restore can be used to restore the original signal after a Workspace-Common or Workspace-Load operation.
- Restore saves a copy of the signal the tool was originally called with. Since saving a copy of long signal can be expensive in terms of memory, the user can turn off this ability. To turn the restore capability off, set the variable `SPC_RESTORE` to *off* in the configuration file `spcolors.m`. See the section on customizing SPC Tools for more information.
- The setting of the `SPC_RESTORE` variable is ignored when the filename option is used to start the tool. This is due to the fact that when the tool is started using a filename as the input argument, only the filename is saved in memory (not the signal) and thus, when Restore is selected, the data file can simply be read in again.

Workspace-Common Menu Item

Purpose

Load the global vector `SPC_COMMON` into the Signal Edit Tool. The `SPC_COMMON` vector is updated by all SPC tools that modify a signal (i.e. editing, filtering, etc.). The Common menu item provides a means of passing signal between tools.

Use

1. Select the Common menu item from the Workspace pull-down menu.

Results

The common signal is loaded.

Notes

- All previous edits to the current signal are lost unless they have been saved using the Workspace-Save menu item.

Workspace-Load Menu Item

Purpose

Load a signal from the MATLAB workspace.

Use

1. Select the Load menu item from the Workspace pull-down menu.
2. A dialog box will open in the tool window.
3. Enter the name of the variable to load into the edit box.
4. Press return or select the pushbutton labeled OK.

Result

The new signal will be loaded.

Notes

- If the variable does not exist in the MATLAB workspace, an error will occur.
- All previous edits to the current signal are lost unless they have been saved using the Workspace-Save menu item.
- To abort loading a new signal, select the Cancel push button in the dialog box.

Workspace-Save Menu Item

Purpose

Save the current edited signal to the MATLAB workspace.

Use

1. Select the Save menu item from the Workspace pull-down menu.
2. A dialog box will open in the tool window.
3. Enter the name used to save the signal into the dialog edit box.
4. Press return or select the pushbutton labeled OK.

Results

The current signal is saved to the MATLAB workspace.

Notes

- If a variable with the same name exists in the MATLAB workspace, it will be overwritten.
- This option is the only way to save a signal which has been edited.

Workspace-File Load Menu Item

Purpose

Load a signal into the tool directly from a data file.

Use

1. Select the File Load menu item to open the file type submenu.
2. Select the file type to load. For Sun audio files, select the Audio (*.au) submenu item. For SoundBlaster Creative Voice files, select the Voice (*.voc) submenu item. For Microsoft Windows wave files, select the Wave (*.wav) submenu item. For Statistical Signal Processing Incorporated time-domain signal files, select the SSPI (*.tim) submenu item. For flat integer data files, select the Integer submenu item and then select the appropriate integer size from the Integer sub-submenu (8-Bits, 16-Bits or 32-Bits).
3. After selecting the appropriate file type, a file dialog box (produced by the MATLAB `uigetfile` command) will appear containing a list of available files. Navigate using the dialog box controls to the desired file and double-click on the filename or select the filename and click on the Done push button. To cancel loading a file, select the Cancel push button in the file dialog box

Results

If a file has been selected, it is read directly into the tool, bypassing the MATLAB workspace. If the Cancel push button is selected, the file dialog is closed and the tool returns to its previous state.

Notes

- The sampling frequency is automatically updated after the file is read. For *.au, *.voc, and *.wav, the sampling frequency is set to that stored in the file. For *.tim or flat integer files, the sampling frequency is set to $fs = 1$. The sampling frequency can be changed using the sampling frequency popup menu as described below.
- Loading signals into the tool directly from data files is more memory efficient and will hence, improve overall performance. The improved performance comes from the facts (1) creation of two copies of the signal is avoided (one in the workspace and one in the tool) and (2) creation of a possible third copy for the restore facility is eliminated since the signal can now be restored directly from the data file if the Workspace-Restore menu item is selected.

Workspace-Quit Menu Item

Purpose

Quit the Signal Edit Tool.

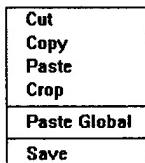
Use

1. Select the Quit menu item from the Workspace pull-down menu.

Results

The Signal Edit Tool is closed. All edits are lost unless previously saved using the Workspace-Save menu item.

Edit Menu



Edit-Cut Menu Item

Purpose

Remove the section of the signal defined by the vertical cursors, placing the cut section into the cut-and-paste buffer.

Use

1. Define the section to be cut using the vertical cursors.
2. Select Cut from the Edit pull-down menu.

Result

The marked section is cut and the display is redrawn.

Notes

- The cut is inclusive (the data points directly under the cursors is included in the cut).
- The removed portion can be restored by immediately selecting Paste from the Edit pull-down menu.
- If the cursors are located at the beginning and end of the signal (i.e. the entire signal is marked for cutting), selecting the Cut push button will have no effect.
- The common signal is modified.

Edit-Copy Menu Item

Purpose

Copy the section of the signal defined by the vertical cursors and place the copied portion into the cut-and-paste buffer.

Use

1. Define the section to be copied using the vertical cursors.
2. Select Copy from the Edit pull-down menu.

Result

The copied section is placed into the cut-and-paste buffer.

Notes

- The copy is inclusive of the data under the cursors.

Edit-Paste Menu Item

Purpose

Insert the contents of the cut-and-paste buffer into the signal at a point selectable with

the mouse.

Use

1. Select Paste from the Edit pull-down menu.
2. The cursor changes into a crosshair and the display title changes to “Mark paste insertion point with cursor.”
3. Move the cursor to the desired insertion point and click the left mouse button.

Result

The contents of the cut-and-paste buffer are inserted into the signal and the signal display is redrawn.

Notes

- If the cut-and-paste buffer is empty, no action is performed (a cut or copy has to be performed first to fill the cut-and-paste buffer).
- The contents of the cut-and-paste buffer are not cleared. Multiple copies of the cut-and-paste buffer can be inserted by reselecting the Paste push button.
- The pasted portion is simply included into the original signal. No transition smoothing is performed.
- The common signal is modified.

Edit-Crop Menu Item

Purpose

Delete portions of the signal that are outside the section delimited by the vertical cursors.

Use

1. Define the section to be retained using the vertical cursors.
2. Select Crop from the Edit pull-down menu.

Result

The signal outside the section delimited by the vertical cursors is deleted.

Notes

- The portion retained is inclusive of the data under the cursors.
- The deleted sections are not saved.
- The cut-and-paste buffer is not modified.
- The common signal is modified.

Edit-Paste Global Menu Item

Purpose

Insert the contents of the global cut-and-paste buffer into the signal at a point selectable with the mouse allowing cut-and-paste operations between Signal Edit Tools.

Use

1. Select Paste Global from the Edit pull-down menu.

2. The cursor changes into a crosshair and the display title changes to "Mark paste insertion point with cursor."
3. Move the cursor to the desired insertion point and click the left mouse button.

Result

The contents of the global cut-and-paste buffer are inserted into the signal and the signal display is redrawn.

Notes

- The contents of the global cut-and-paste buffer is that of the last cut or copy operation in any sigedit tool.
- All notes under Edit-Paste apply here.

Edit-Save Menu Item

Purpose

Save the section of the signal defined by the vertical cursors to a MATLAB workspace variable.

Use

1. Define the section to be saved using the vertical cursors.
2. Select Save from the Edit pull-down menu.
3. A dialog box will open in the tool window.
4. Enter the name to save the section under into the dialog edit box.
5. Press return or select the pushbutton labeled OK.

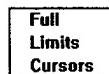
Results

The signal section is saved to the MATLAB workspace.

Notes

- If a variable with the same name already exists in the MATLAB workspace, it will be overwritten.

Play Menu



Play-Full Menu Item

Purpose

Send the signal to the workstation's audio output.

Use

1. Select Full from the Play pull-down menu.

Result

The entire signal is sent to the workstation's audio output.

Notes

- The signal is played at the sampling frequency set on the Sampling Freq popup menu on workstations supporting multiple output sampling frequencies.

Play-Limits Menu Item

Purpose

Send the signal displayed within the axis limits to the workstation's audio output.

Use

1. Use the zoom controls to zoom into the section of the signal to be played.
2. Select Limits from the Play pull-down menu.

Result

The signal displayed within the axis limits is sent to the workstation's audio output.

Notes

- The signal is played at the sampling frequency set on the Sampling Freq popup menu on workstations supporting multiple output sampling frequencies.

Play-Cursors Menu Item

Purpose

Send the signal displayed between the vertical cursors to the workstation's audio output.

Use

1. Define the section to be played using the vertical cursors.
2. Select Cursors from the Play pull-down menu.

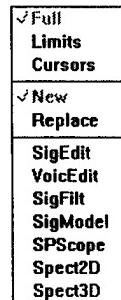
Result

The signal displayed between cursors is sent to the workstation's audio output.

Notes

- The signal is played at the sampling frequency set on the Sampling Freq popup menu on workstations supporting multiple output sampling frequencies.

Launch Menu



Purpose

Launch additional SPC Toolbox tools containing all or a portion of the current signal.

Use

1. Check the menu item corresponding to the signal portion to be used (Full, Limits or Cursors).
2. Check the menu item corresponding to launching a New tool or to Replace a previously opened tool of the same type (to be selected in step three).
3. Select the SPC Toolbox tool to launch.

Result

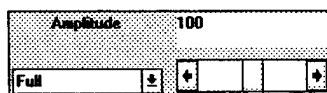
A new tool is launched containing the desired signal. The setting of the Sampling Freq popup menu is automatically passed to the new tool (if required).

Notes

- This option provides a quick and easy method for processing portions of a signal without having to go through a crop-save-load cycle.
- The number of additional tools that can be launched is limited only by the workstation capabilities (i.e. available memory).

Signal Edit Tool Controls

Amplitude Control Group



Purpose

Adjust the amplitude (volume) of the signal or a vertical cursor delimited section 0% to 200% of its present amplitude.

Use

1. Define the section to be adjusted using the vertical cursors (if desired).
2. Adjust the volume slider to the desired amplitude change percentage or enter the desired percentage into the edit box.
3. Place the cursor on the Amplitude popup menu and click the left mouse button once to open the popup menu.
4. Place the cursor on the desired function (Full or Marked) and click the left mouse button.

Result

The amplitude is adjusted as requested and the display is redrawn.

Notes

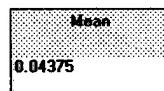
- If the signal is to be saved in an audio file format, care must be taken to ensure the new range of values does not exceed the limits of the target format. For example, Microsoft Window "wave" files and Soundblaster "voice" files often contain only 8-bit values (-128 to +127). Values outside this range are clipped when saved to the

audio file. The amplitude can be properly adjusted to range between +127 and -127 by the MATLAB command

```
>> mysignal = 127 * mysignal / max(mysignal);
```

- The common signal is modified.

Mean User Edit Box



Purpose

- Display the current signal mean value.
- Remove the mean.
- Change the mean (if the signal represents a voltage, this is equivalent to changing the DC offset).

Use

Removing the mean

1. Select the edit box by clicking the left mouse button while the cursor is located over the edit box.
2. Enter zero into the edit box and press return.

Changing the mean

1. Select the edit box by clicking the left mouse button while the cursor is located over the edit box.
2. Enter the desired offset value into the edit box and press return.

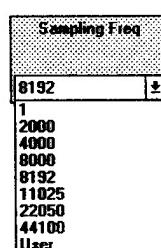
Result

The mean is adjusted and the display is redrawn.

Notes

- After the mean is changed, the mean is recomputed and redisplayed in the edit box. Note that after removing the signal mean, the resulting value may not be exactly zero due to machine round-off errors.
- The common signal is modified.

Sampling Freq Popup Menu



Purpose

Set the sampling frequency.

Use

1. Place the cursor on the Sampling Freq popup menu and click the left mouse button once to open the menu.
2. Select the sampling frequency with the left mouse button. If the sampling frequency is not available on the menu, the desired sampling frequency can be entered by selecting the User menu item. After User is selected, an edit box appears in place of the popup menu. Enter the desired sampling frequency into this edit box and press return. The value entered will now be listed on the popupmenu.

Result

The display is redrawn with a new time axis according to the sampling frequency.

Notes

- If the sampling frequency is set to 1 Hz, the horizontal axis scale will correspond to the vector indices.

Zero Marked Push Button**Purpose**

Zero the section of the signal defined by the vertical cursors. This option is useful for creating periods of silence.

Use

1. Define the section to be zeroed out using the vertical cursors.
2. Place the cursor on the Zero Marked push button and click the left mouse button.

Result

The signal between the cursors is set to zero and the display is redrawn.

Notes

- The block zeroed is inclusive of the cursor data points.
- The common signal is modified.

Examples

Load the speech signal contained in the audio file `seatsit.voc`.

```
s = loadvoc('seatsit');
sigedit(s)
```

Cut and paste operation may now proceed.

Related Files

- `sigedit.m` - Loads signal from workspace.
- `sigedsav.m` - Saves edited signal to workspace.

sigedit

- `sigedit.m` - Callback functions for the Signal Edit Tool.

See Also

`voicedit`, `zoomtool`

sigfilt

Purpose

Analog-prototype, digital-filtering tool.

Synopsis

```
sigfilt(x)
sigfilt(x,fs)
sigfilt('filename.ext')
sigfilt('filename',bits)
h = sigfilt(x,fs,callback,parent);
```

Description

`sigfilt(x)` opens a signal filtering tool containing the time-domain signal `x`.

`sigfilt(x,fs)` sets the sampling frequency to `fs`. The default sampling frequency is 8192 Hz. A sampling frequency of 1 Hz will display the time axis in sample number vice time.

`sigfilt('filename.ext')` - Opens the file specified by '`filename.ext`' and loads the signal stored within the file directly into the tool. File formats with the extensions `*.au`, `*.voc`, `*.wav`, and `*.tim` can be specified by the '`filename.ext`' option only (be sure to include the extension). Path names are required when the file is not in the current directory. The sampling frequency is set to that stored in the file for `*.au`, `*.voc` and `*.wav` formats and to `fs = 1 Hz` for `*.tim` files. See the function `readsig` for more information on the functions used to read signal files.

`sigfilt('filename',bits)` - Opens the file specified by '`filename`' and loads the signal stored within directly into the tool as a flat integer formatted file. The number of bits-per-integer is specified by the `bits` argument where `bits` is either 8, 16, or 32. Integer formats must be stored as signed integers to be read correctly. Multi-byte integers (16 and 32) are assumed to be compatible with the format used by the workstation in use ("Little Endian" and "Big Endian" byte ordered files are not compatible). Path names are required when the file is not in the current directory. The sampling frequency is set according to the file type loaded. See the File-Load menu item description for more information.

`h = sigfilt(x,fs,callback,parent)` returns the handle to the figure window `sigfilt` was opened in. This use supports communication between different SPC Toolbox tools and is useful for keeping track of `sigfilt` when it is called by a parent tool. The `callback` function is executed after the user selects the Apply-Use menu item which is enabled after the filter has been applied at least once using the Apply-Test menu item. This should normally be used to force the calling tool to load the global `SPC_COMMON` variable. Since this variable is updated each time the filter is applied, forcing a tool to load this variable has the same effect as sending the filtered signal back to the calling tool. An example is when `sigfilt` is launched from `sigedit` using the Filter-Analog menu item. The `parent` argument is the figure handle of the calling tool. See the `zoomprog.m` file for programming hints.

Note: No output arguments are supported. Edited signals must be saved using the Save menu item under the Workspace pull-down menu.

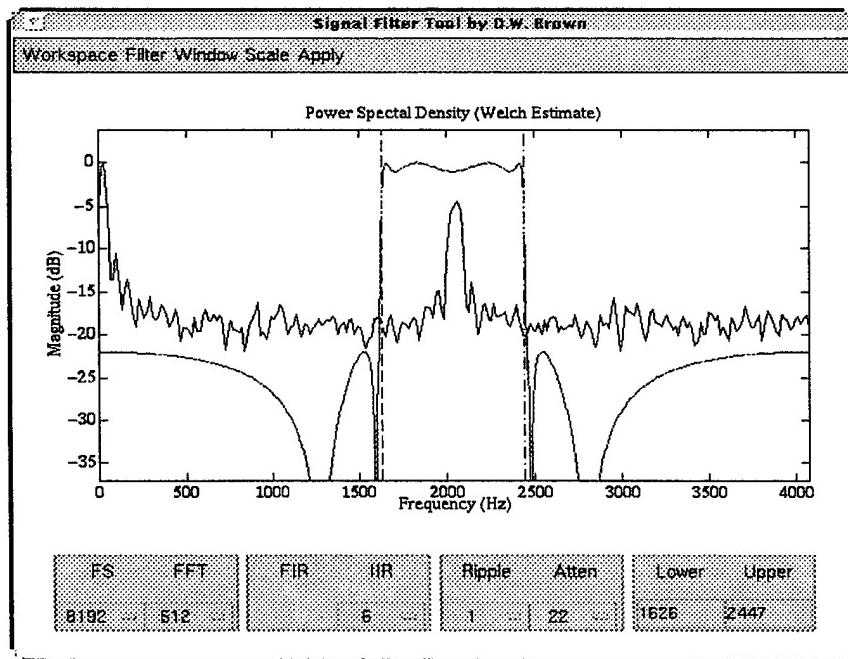
Signal Filter Tool

The Signal Filter Tool provides an interactive, graphical environment within MATLAB to filter vectors. This is ideally suited for filtering vectors that represent digitized signals such as digitized speech or communications signals. With the Signal Filter Tool, you can:

- Perform lowpass, highpass, bandpass and stopband filtering.
- Filter using Finite Impulse Response (FIR) filters.
- Filter using Butterworth, Chebychev or Elliptical Infinite Impulse Response (IIR) filters.
- Interactively set filter parameters such as FIR window, filter order and stopband attenuation.
- View the filter transfer function before applying.
- Determine the audible effects of filtering by playing filtered signals on the workstation audio output.

Signal Filter Tool Screen

The Signal Filtering Tool opens displaying a Welch spectral estimate of the signal in the main tool window and the signal itself in a separate Time-Domain Signal window. In this time-domain window, zoomtool provides a means to inspect the signal and to take oscilloscope type measurements. The frequency-domain also contains the transfer function of the currently designed filter. A cursor marks the location of the lower cut-off frequency used in generating the filter transfer function. This capability to display the transfer function against the backdrop of the signal spectrum allows the filter designer to fine tune the filter to the response required by the application. To best show the spectrum with regard to the transfer function on a logarithmic scale, the entire signal spectrum is shifted up or down so that the peak value aligns with zero dB. Thus displayed, the logarithmic scale can only be used for taking relative power measurements (i.e. zero dB does not necessarily correspond to 1 Watt on the linear scale).



Dependent upon the current filter type under design (lowpass, highpass, bandpass or stopband), one or two cursors will appear in the frequency-domain axis. When a bandpass or stopband filter is being designed, the second cursor indicates the upper cutoff frequency. Either cursors can be "grabbed" with the mouse and moved to a new location in order to change the corresponding cutoff frequency. They can also be moved by entering values into the Lower or Upper cutoff frequency edit boxes as discussed below. Moving one cursor past the other will change both the upper and lower cutoff frequencies when designing bandpass and stopband filters.

A number of controls located under the graphs are used to set the filter parameters in conjunction with the Filter and Window pull-down menus. Only valid parameters for the current design are enabled throughout the design process. For example, if the current filter type under design is a lowpass filter, there is no need to set an upper cutoff frequency so the Upper cutoff frequency edit box is not displayed. Changing the filter type to bandpass or stopband will turn on display of the Upper cutoff frequency edit box since it is then a valid parameter for the current filter.

As filter design progresses, the transfer function display is updated to reflect the frequency response of the current filter design. Normally, the magnitude axis is scaled to maximize the view of the signal spectrum. This scale is usually not large enough to display the entire transfer function due to the fact the transfer function normally has large valued negative nulls (when on a logarithmic scale). To view the entire transfer function, select the Full Magnitude menu item from the Scale pull-down menu.

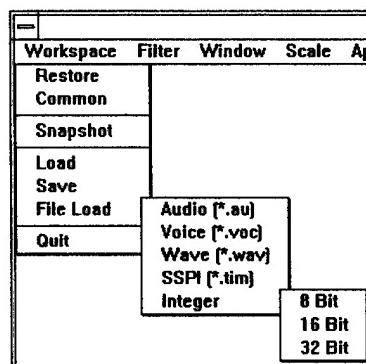
After selecting Full Magnitude, the detail in the frequency magnitude curve may not be discernible. To zoom back in on the frequency curve, the mouse pointer is used. With the mouse pointer in a blank region of the axis, pressing the left mouse button resets the top or bottom axis limit. On a logarithmic scale, pressing the left mouse button with the mouse pointer below the zero decibels resets the bottom axis limit at the

mouse pointer location. Similarly, pressing the left mouse button with the mouse pointer above the zero decibels resets the top axis limit. On a linear scale, pressing the left mouse button resets the top axis limit in all cases. When resetting the top and bottom magnitude limits in this manner, the mouse pointer must be in a blank screen region inside the axis and not near any lines or the axis limits themselves.

The filter is applied to the signal using the Apply-Filter pull-down menu item. After the signal has been filtered, the spectrum is recomputed and displayed along with the transfer function and the previous spectrum drawn as a dotted line. This allows the user to actually see how effective the filter was at attenuating unwanted signals.

Signal Filter Tool Pull-Down Menus

Workspace Menu



Workspace-Restore Menu Item

Purpose

Restore the original signal.

Use

1. Select the Restore menu item from the Workspace pull-down menu.

Results

The original signal is restored.

Notes

- Restore can be used to restore the original signal after a Workspace-Common or Workspace-Load operation.
- Restore saves a copy of the signal the tool was originally called with. Since saving a copy of long signal can be expensive in terms of memory, the user can turn off this ability. To turn the restore capability off, set the variable SPC_RESTORE to *off* in the configuration file spcolors.m. See the section on customizing SPC Tools for more information.
- The setting of the SPC_RESTORE variable is ignored when the filename option is used to start the tool. This is due to the fact that when the tool is started using a filename as the input argument, only the filename is saved in memory (not the signal) and thus, when Restore is selected, the data file can simply be read in again.

Workspace-Common Menu Item

Purpose

Loads the global vector SPC_COMMON into the signal filter tool. The SPC_COMMON vector is updated by all SPC tools that modify a signal (i.e. editing, filtering, etc.). The Common menu item provides a means of passing signal between tools.

Use

1. Select the Common menu item from the Workspace pull-down menu.

Results

The common signal is loaded.

Notes

- Filtered signals are lost unless they have been saved using the Workspace-Save menu item.

Workspace-Load Menu Item

Purpose

Load a signal from the MATLAB workspace.

Use

1. Select the Load menu item from the Workspace pull-down menu.
2. A dialog box will open in the tool window.
3. Enter the name of the variable to load into the edit box.
4. Press return or select the pushbutton labeled OK.

Result

The new signal will be loaded.

Notes

- If the variable does not exist in the MATLAB workspace, an error will occur.
- Filtered signals are lost unless they have been saved using the Workspace-Save menu item.
- To abort loading a new signal, select the Cancel push button in the dialog box.

Workspace-Save Menu Item

Purpose

Save the current edited signal to the MATLAB workspace.

Use

1. Select the Save menu item from the Workspace pull-down menu.
2. A dialog box will open in the tool window.
3. Enter the name to save the signal under into the dialog edit box.
4. Press return or select the pushbutton labeled OK.

Results

The current signal is saved to the MATLAB workspace.

Notes

- If a variable with the same name exists in the MATLAB workspace, it will be overwritten.
- This is the only way to save a signal that has been filtered.

Workspace-File Load Menu Item

Purpose

Load a signal into the tool directly from a data file.

Use

1. Select the File Load menu item to open the file type submenu.
2. Select the file type to load. For Sun audio files, select the Audio (*.au) submenu item. For SoundBlaster Creative Voice files, select the Voice (*.voc) submenu item. For Microsoft Windows wave files, select the Wave (*.wav) submenu item. For Statistical Signal Processing Incorporated time-domain signal files, select the SSPI (*.tim) submenu item. For flat integer data files, select the Integer submenu item and then select the appropriate integer size from the Integer sub-submenu (8-Bits, 16-Bits or 32-Bits).
3. After selecting the appropriate file type, a file dialog box (produced by the MATLAB `uigetfile` command) will appear containing a list of available files. Navigate using the dialog box controls to the desired file and double-click on the filename or select the filename and click on the Done push button. To cancel loading a file, select the Cancel push button in the file dialog box

Results

If a file has been selected, it is read directly into the tool, bypassing the MATLAB workspace. If the Cancel push button is selected, the file dialog is closed and the tool returns to its previous state.

Notes

- The sampling frequency is automatically updated after the file is read. For *.au, *.voc, and *.wav, the sampling frequency is set to that stored in the file. For *.tim or flat integer files, the sampling frequency is set to $fs = 1$. The sampling frequency can be changed using the sampling frequency popup menu as described below.
- Loading signals into the tool directly from data files is more memory efficient and will hence, improve overall performance. The improved performance comes from the facts (1) creation of two copies of the signal is avoided (one in the workspace and one in the tool) and (2) creation of a possible third copy for the restore facility is eliminated since the signal can now be restored directly from the data file if the Workspace-Restore menu item is selected.

Workspace-Snapshot Menu Item

Purpose

Make a copy of the frequency axis in a new figure window.

Use

1. Select the Snapshot menu item from the Workspace pull-down menu.

Results

The current frequency axis display is copied to a new figure window.

Notes

- This option is handy for obtaining hardcopy of the current spectrum. Once the new figure is created it can be printed or otherwise manipulated (e.g. change labels, add title or text, etc.).

Workspace-Quit Menu Item

Purpose

Quit the signal filter tool.

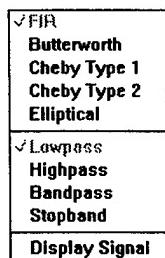
Use

1. Select the Quit menu item from the Workspace pull-down menu.

Results

The signal filter tool is closed. Filtered signals are lost unless previously saved using the Workspace-Save menu item.

Filter Menu



Filter Pull-Down Menu

Purpose

Select the analog filter prototype to use and the filter characteristic type.

Use

To select the analog prototype filter:

1. Place the cursor on the window popup menu and click the left mouse button once to open.
2. Select the desired analog prototype (FIR, Butterworth, Chebychev Type 1, Chebychev

Type 2, or Elliptical) from the Filter pull-down menu.

To select the filter type:

1. Place the cursor on the window popup menu and click the left mouse button once to open.
2. Select the desired type (lowpass, highpass, bandpass, or stopband) from the Filter pull-down menu.

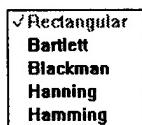
Result

The desired filtering method or type menu item is checked and the transfer function is recomputed and displayed. Additional controls are added or removed dependent upon the filtering method chosen.

Notes

The filters are implemented using functions from the MATLAB *Signal Processing Toolbox* which is required to use this tool.

FIR Window Popup Menu



Purpose

Selects the type of smoothing window used in computing the filter coefficients of an FIR filter.

Use

1. Place the cursor on the window popup menu and click the left mouse button once to open.
2. Place the cursor on the desired window (Rectangular, Blackman, Bartlett, Hamming, or Hanning) and click the left mouse button.

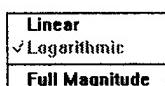
Result

The desired window is selected and displayed on the popup menu.

Notes

- The windows are implemented using functions from the MATLAB *Signal Processing Toolbox* which is required to use this tool.
- This control is available only when the current filtering method is set to FIR.

Scale Menu



Purpose

Select the scaling of the frequency magnitude plot (logarithmic or linear).

Use

1. Pull-down the Scale menu and select the desired Linear or Logarithmic menu item.

Result

The menu-item of the selected scale is checked. The scale is immediately applied to the Welch estimate contained in the tool. The window will be applied to the next spectral surface generated.

Scale-Full Magnitude Menu Item**Purpose**

Adjust frequency domain magnitude scaling so that the entire signal spectrum and transfer function is displayed.

Use

1. Select the Full Magnitude menu item from the Scale pull-down menu.

Result

The frequency-domain axis is redrawn to show the entire signal spectrum and transfer function.

Apply Menu**Apply-Filter Pushbutton****Purpose**

Applies the filter.

Use

1. Select the Filter menu item from the Apply pull-down menu.

Result

The filter specified by the parameter controls is applied to the vector and the result displayed.

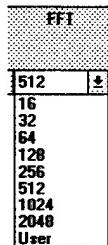
Notes

- The result of an unstable filter should be immediately recognizable. Select Workspace-Restore to return the original signal and try increasing the filter order.

Signal Filter Tool Specific Controls

For the most part, these controls simply set the filter parameters. No filtering actually takes place until the Apply-Filter menu item is selected. Additionally, not all of the following controls are available at any one time. Only controls available to the currently selected filter and filter type are displayed. For example, there is no need to set an upper cutoff frequency when a low or highpass filter type is selected.

FFT Length Popup Menu



Purpose

Set the FFT length used in the spectrum display.

Use

1. Place the cursor on the FFT popup menu and click the left mouse button once to open the menu.
2. Place the cursor on the desired FFT length and click the left mouse button. If the desire FFT length is not available on the menu, it can be entered by selecting the User menu item. After User is selected, an edit box appears in place of the popup menu. Enter the desired FFT length into this edit box and press return. The entered value is now listed on the popup menu.

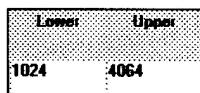
Result

The spectrum is recomputed and displayed.

Notes

- In the popup menu, the User menu item is replaced by the FFT length entered using the edit box. To subsequently change the FFT length after previously using the edit box, the last FFT length on the popup menu has to be selected (this is where the User menu item was).
- Only the positive frequencies are displayed. Thus, out of a 512 point FFT, only 256 points are displayed.
- The frequency resolution between points is fs/FFT_length .

Lower Cutoff Frequency Edit Box



Purpose

- Display the lower cutoff frequency when it is selected using the mouse.
- Allow the operator to manually enter the lower cutoff frequency.

Use

When selecting by mouse:

1. Display the spectrum of the signal using the Spectrum radiobutton.
2. Mark the lower cutoff frequency using the beginning mark.

Manual entry:

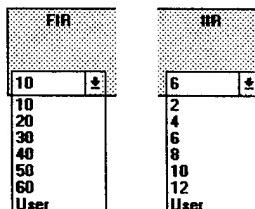
1. Place the cursor over the edit box and click the left mouse button.
2. Enter the desired lower cutoff frequency and press return.

Result

The lower cutoff frequency is displayed in the edit box.

Notes

- If the lower cutoff frequency is (1) less than or equal to zero or (2) greater than or equal to $f_s/2$ or the upper cutoff frequency, the error message “Invalid lower cutoff frequency entered. Try again...” is displayed in a dialog box (if supported) or displayed in the MATLAB command window. If your version of MATLAB supports dialog boxes, acknowledge the OK push button. The edit box will be cleared and the lower cutoff frequency reentered. This cycle continues until a valid lower cutoff frequency is entered.
- After a cutoff frequency has been set, the transfer function of the currently defined filter is drawn against the background of the signal spectrum.
- Setting the lower cutoff frequency too close to zero, $f_s/2$ or the upper cutoff frequency can lead to an unstable filter.
- The effect of an unstable filter is immediately recognizable after applying the filter.

FIR and IIR Order Popup Menus**Purpose**

Set the filter order.

Use

1. Place the cursor on the FIR or IIR popup menu and click the left mouse button once to open the menu.
2. Place the cursor on the desired filter order and click the left mouse button. The popup menu contains orders of 11, 21, 31, 41, 51 and 61 when the FIR filter is selected and orders of 2, 4, 6, 8, 10 and 12 when an IIR filter is selected. If the desired order is not available on the menu, it can be entered by selecting the User menu item. After User is selected, an edit box appears in place of the popup menu. Enter the desired order into this edit box and press return. The entered value is now listed on the popup menu.

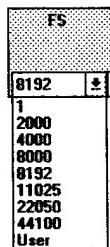
Result

The filter order is set.

Notes

- The filter order for bandpass and stopband filters is twice the order shown on the popup menu. This is consistent with the input argument specifications for the MATLAB *Signal Processing Toolbox* routines.
- Highpass and stopband filters require an even order. Odd orders are automatically rounded up one to the next higher even order.
- The filters are implemented using functions from the MATLAB *Signal Processing Toolbox* which is required to use this tool. Some restrictions on the filter order apply when using these functions. See the MATLAB *Signal Processing Toolbox* documentation for specifics.
- In general, FIR filters require a high order to obtain a sharp rolloff. IIR filters can achieve the same or better results using a much lower order.

Sampling Freq Popup Menu



Purpose

Set the sampling frequency.

Use

1. Place the cursor on the Sampling Freq popup menu and click the left mouse button once to open the menu.
2. Select the sampling frequency with the left mouse button. If the sampling frequency is not available on the menu, the desired sampling frequency can be entered by selecting the User menu item. After User is selected, an edit box appears in place of the popup menu. Enter the desired sampling frequency into this edit box and press return. The entered value is now listed on the popup menu.

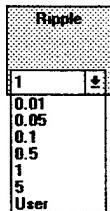
Result

The display is redrawn with a new time axis according to the sampling frequency.

Notes

- If the sampling frequency is set to 1 Hz, the time-axis scale will correspond to the vector indices.

Passband Ripple Popup Menu



Purpose

Set the amount of allowable ripple in the passband of Chebychev or elliptical filters.

Use

1. Place the cursor on the PB popup menu and click the left mouse button once to open the menu.
2. Place the cursor on the desired passband ripple and click the left mouse button. The popup menu contains ripple values of 0.01, 0.05, 0.1, 0.5, 1.0 and 5.0 decibels. If the desired order is not available on the menu, it can be entered by selecting the User menu item. After User is selected, an edit box appears in place of the popup menu. Enter the desired order into this edit box and press return. The value entered is now displayed on the popup menu.

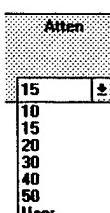
Result

The passband ripple is set.

Notes

- This control is available only when the current filtering method is set to Chebychev Type 1 or Elliptical.

Stopband Attenuation



Purpose

Sets the amount frequencies in the stopband are attenuated.

Use

1. Place the cursor on the Atten popup menu and click the left mouse button once to open the menu.
2. Place the cursor on the desired stopband attenuation and click the left mouse button. The popup menu contains attenuation values of 10, 15, 20, 30, 40 and 50 dB decibels. If the desired order is not available on the menu, it can be entered by selecting the User menu item. After User is selected, an edit box appears in place of the

popup menu. Enter the desired order into this edit box and press return.

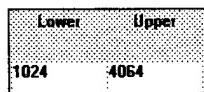
Result

The stopband attenuation is set.

Notes

- This control is available only when the current filtering method is set to Chebychev Type 2 or Elliptical.

Upper Cutoff Frequency Edit Box



Purpose

- Display the upper cutoff frequency when it is selected using the mouse.
- Allow the operator to manually enter the upper cutoff frequency.

Use

When selecting by mouse:

1. Display the spectrum of the signal using the Spectrum radiobutton.
2. Mark the upper cutoff frequency using the end mark.

Manual entry:

1. Place the cursor over the edit box and click the left mouse button.
2. Enter the desired upper cutoff frequency and press return.

Result

The upper cutoff frequency is displayed in the edit box.

Notes

- If the upper cutoff frequency is (1) less than or equal to zero or (2) greater than or equal to $f_s/2$ or the lower cutoff frequency, the error message “Invalid upper cutoff frequency entered. Try again...” is displayed in a dialog box (if supported) or displayed in the MATLAB command window. If your version of MATLAB supports dialog boxes, acknowledge the OK push button. The edit box will be cleared and the upper cutoff frequency reentered. This cycle continues until a valid upper cutoff frequency is entered.
- Setting the upper cutoff frequency too close to zero, $f_s/2$ or the lower cutoff frequency can lead to an unstable filter.
- The effect of an unstable filter is immediately recognizable after applying the filter.

Examples

Use *sigfilt* to demodulate a BPSK signal. First generate a one-second long, randomly keyed 64 bps BPSK signal at a carrier frequency of 1024 Hz and a sampling frequency of 8192 Hz. Mix the signal with Gaussian white noise such that the noise-equivalent bandwidth is 15 dB. Multiply the simulated “received” signal with an in-phase sinu-

soid to create images of the signal at baseband and at twice the carrier frequency.

```
s = bpsk(64, 1024);
x = setsnrbw(s,15,1024,64);
y = x .* coswave(1024);
sigfilt(y)
```

Design a lowpass filter and apply. The demodulated signal will appear in the time-domain window.

Related Files

- *sigfld.m* - Loads signal from workspace.
- *sigfisav.m* - Saves filtered signal to workspace.
- *sigfecal.m* - Callback functions for the Signal Filter Tool.

See Also

zoomtool

sigmodel

Purpose

Signal model design tool.

Synopsis

```
sigmodel(x);  
sigmodel(x,fs);  
sigmodel('filename.ext')  
sigmodel('filename',bits)  
h = sigmodel(x,fs);
```

Description

`sigmodel(x)` opens a Signal Modeling Tool containing the time-domain signal `x`.

`sigmodel(x,fs)` sets the sampling frequency to `fs`. The default sampling frequency is 8192 Hz. A sampling frequency of 1 Hz will display the time axis in sample number versus time.

`sigmodel('filename.ext')` - Opens the file specified by '`filename.ext`' and loads the signal stored within the file directly into the tool. File formats with the extensions `*.au`, `*.voc`, `*.wav`, and `*.tim` can be specified by the '`filename.ext`' option only (be sure to include the extension). Path names are required when the file is not in the current directory. The sampling frequency is set to that stored in the file for `*.au`, `*.voc` and `*.wav` formats and to `fs = 1 Hz` for `*.tim` files. See the function `readsig` for more information on the functions used to read signal files.

`sigmodel('filename',bits)` - Opens the file specified by '`filename`' and loads the signal stored within directly into the tool as a flat integer formatted file. The number of bits-per-integer is specified by the `bits` argument where `bits` is either 8, 16, or 32. Integer formats must be stored as signed integers to be read correctly. Multi-byte integers (16 and 32) are assumed to be compatible with the format used by the workstation in use ("Little Endian" and "Big Endian" byte ordered files are not compatible). Path names are required when the file is not in the current directory. The sampling frequency is set according to the file type loaded. See the File-Load menu item description for more information.

`h = sigmodel(x,fs)` returns the handle to the figure window it was opened in. This is useful for programmers to keep track of `sigmodel` when it is called by a parent application. Such an example occurs when `sigmodel` is launched from `voicedit`. See `zoomprog.m` for programming hints.

Note: No output arguments are supported. Model coefficients and the model and error signals must be saved using the save capabilities in the individual model design tool dialog boxes.

Signal Model Tool

The Signal Model Tool provides an interactive, graphical tool from which to model time-domain signals. Using this tool, the user can designate a portion of the signal

using movable cursors or by entering start and stop times of the data to be included in deriving the model parameters. The user can also designate a period length (the length of the model generated) in a similar manner. This ability to designate a separate model data length and a model period length is quite useful when developing a model for a single pitch period using data from several pitch periods in speech signals. In this case, the Model Data cursors would delineate several pitch periods and the Period cursors would delineate the length of a single pitch period.

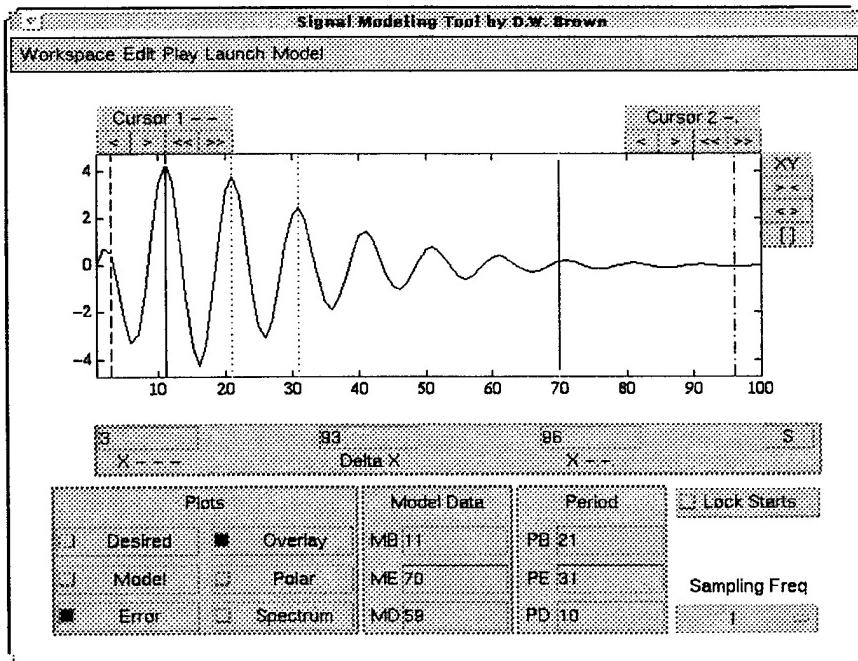
Additional controls in the Signal Model Tool allow the user to designate a number of different graphic plots. These include plots of the model and error signals, a polar pole-zero plot of the model transfer function roots and a spectral estimate plot when the model being derived can be used in spectrum estimation. Like for all SPC Toolbox time-domain tools, a Launch menu is provided from which the user can start additional SPC tools containing all or a portion of the current signal in the Signal Model Tool.

Selecting a modeling method from the Model pull-down menu opens dialog boxes in which the user sets particular parameters for the model under development. After the Apply push button has been selected in a model design dialog box, the desired model is generated and any requested plots are created. To save models and model parameters to the MATLAB workspace, controls in the model design dialog box are used. See the particular model design tool for more information about controls in the model dialog boxes (i.e. armadsgn).

Signal Model Tool Screen

The Signal Model Tool screen consists of a set of pull-down menus and a group of controls along with an axis containing the signal to be modeled. This axis uses zoomtool to provide zoom capabilities along with cursor controls and cursor position read-outs. Only the vertical cursors of zoomtool are displayed. See the zoomtool description for more information on cursor movement and zoom operations.

In addition to the zoomtool cursors, two additional pairs of vertical cursors are located in the time-domain axis. These cursors are used to designate the data to be used in generating the model (Model Data) and the model length (Period). All cursors are located at the axis limits when sigmodel is initially started. In this position, the cursors are stacked on top of each other and are thus hard to individually "grab" and move with the mouse. Fortunately, the zoomtool cursors can be easily grabbed by simply pressing the left mouse button with the mouse pointer above any point on the time-domain signal near the zoomtool cursors. To enable grabbing the model and period cursors with the mouse first requires moving these cursors away from the others. This is accomplished by entering discrete locations into the Model Data group and Period edit boxes. Once a value has been entered into one of these edit boxes and the return key pressed, the affected cursor is moved to the new locations. As long as this new location does not coincide with another cursor, the affected cursor can now be grabbed with the mouse pointer and moved to the desired location.



The Signal Model Tool also has all the functionality of the Signal Edit Tool in the ability to cut and paste edit the signal. This capability lets the user cut a small portion from a much larger signal, effectively speeding up the operation of sigmodel.

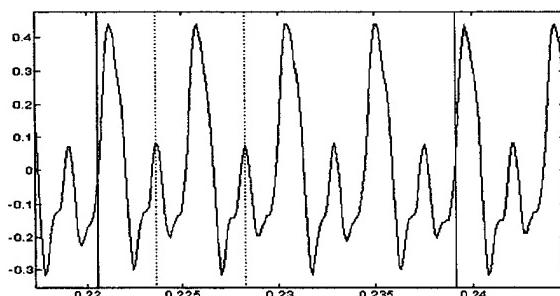
Model Setup Procedures

The Model Data and Period edit box groups both contain three edit boxes. The first edit box (MB or PB) designates the location of the model or period beginning. The second edit box (ME or PE) designates the location of the model or period end. The third edit box (MD or PD) is the difference between the beginning and end. Setting up to generate a model begins by setting the model and period lengths using these edit boxes or by moving the model (solid) and period (dotted) cursors. In general, there are four scenarios possible dealing with the model data and period lengths. These are outlined below along with the steps required to set up each scenario.

1. All available data is used in generating the model and the length of the model (period) equals the length of the signal being modeled.
 - Leave the model and period cursors in their initial positions at the axis limits.
2. The length of data used in generating the model and the length of the model (period) are equal but shorter than the signal being modeled.
 - Ensure the Lock Starts check box is checked. A checked Lock Starts check box causes the location of the MB and PB cursors to always coincide.
 - Move either the MB or PB cursor to the beginning of the data section to use in generating the model (the other will automatically follow).
 - Move both the ME and PE cursors to the same location (these have to be moved individually).
3. The length of data used in generating the model is shorter than the length of the

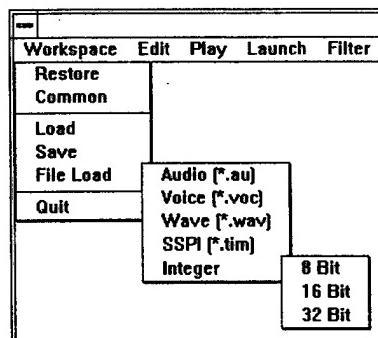
model being generated. This situation arises when an accurate model can be derived from just a small portion of the data.

- Proceed as in situation two above only place the ME cursor closer to the MB cursor than the PE cursor is to the PB cursor.
4. The length of data used in generating the model is longer than the length of the model being generated. This situation is typical when modeling speech signals in which obtaining the model parameters for one pitch period requires using data from several pitch periods.
 - Proceed as in scenario two above, only placing the PE cursor one pitch period before the PB cursor, and place several pitch periods between the MB and ME cursors.
 - Dependent upon the modeling method, a better model may result in generating the model beginning from a zero crossing point in the data. However, setting the length of the pitch period may be more accurately performed using peaks in the signal. In this case, unlock the Lock Starts check box and independently set the model and period lengths. Note, only the length designated by the separation of the period markers is used, not the data contained between the two. The following figure illustrates a situation where data contained in four pitch periods is used to model one pitch period of a voiced phoneme. The data used to generate the model begins at a zero crossing and the length of the model is determined by the length between two consecutive smaller peaks in the signal.



Signal Model Tool Pull-Down Menus

Workspace Menu



Workspace-Restore Menu Item

Purpose

Restore the original signal.

Use

1. Select the Restore menu item from the Workspace pull-down menu.

Results

The original signal is restored.

Notes

- Restore can be used to restore the original signal after a Workspace-Common or Workspace-Load operation.
- Restore saves a copy of the signal the tool was originally called with. Since saving a copy of long signal can be expensive in terms of memory, the user can turn off this ability. To turn the restore capability off, set the variable SPC_RESTORE to *off* in the configuration file spcolors.m. See the section on customizing SPC Tools for more information.
- The setting of the SPC_RESTORE variable is ignored when the filename option is used to start the tool. This is due to the fact that when the tool is started using a filename as the input argument, only the filename is saved in memory (not the signal) and thus, when Restore is selected, the data file can simply be read in again.

Workspace-Common Menu Item

Purpose

Load the global vector SPC_COMMON into the Signal Model Tool. The SPC_COMMON vector is updated by all SPC tools that modify a signal (i.e. editing, filtering, etc.). The Common menu item provides a means of passing signal between tools.

Use

1. Select the Common menu item from the Workspace pull-down menu.

Results

The common signal is loaded.

Notes

- All previous edits to the current signal are lost unless they have been saved using the Workspace-Save menu item.

Workspace-Load Menu Item

Purpose

Load a signal from the MATLAB workspace.

Use

1. Select the Load menu item from the Workspace pull-down menu.
2. A dialog box will open in the tool window.
3. Enter the name of the variable to load into the edit box.
4. Press return or select the pushbutton labeled OK.

Result

The new signal will be loaded.

Notes

- If the variable does not exist in the MATLAB workspace, an error will occur.
- All previous edits to the current signal are lost unless they have been saved using the Workspace-Save menu item.
- To abort loading a new signal, select the Cancel push button in the dialog box.

Workspace-Save Menu Item

Purpose

Save the current edited signal to the MATLAB workspace.

Use

1. Select the Save menu item from the Workspace pull-down menu.
2. A dialog box will open in the tool window.
3. Enter the name used to save the signal into the dialog edit box.
4. Press return or select the pushbutton labeled OK.

Results

The current signal is saved to the MATLAB workspace.

Notes

- If a variable with the same name exists in the MATLAB workspace, it will be overwritten.
- This option is the only way to save a signal which has been edited.
- The model coefficients, model signal, and error signal must be saved using the modeling method dialog box.

Workspace-File Load Menu Item

Purpose

Load a signal into the tool directly from a data file.

Use

1. Select the File Load menu item to open the file type submenu.
2. Select the file type to load. For Sun audio files, select the Audio (*.au) submenu item. For SoundBlaster Creative Voice files, select the Voice (*.voc) submenu item. For Microsoft Windows wave files, select the Wave (*.wav) submenu item. For Statistical Signal Processing Incorporated time-domain signal files, select the SSPI (*.tim) submenu item. For flat integer data files, select the Integer submenu item and then select the appropriate integer size from the Integer sub-submenu (8-Bits, 16-Bits or 32-Bits).
3. After selecting the appropriate file type, a file dialog box (produced by the MATLAB `uigetfile` command) will appear containing a list of available files. Navigate using the dialog box controls to the desired file and double-click on the filename or select the filename and click on the Done push button. To cancel loading a file, select the Cancel push button in the file dialog box

Results

If a file has been selected, it is read directly into the tool, bypassing the MATLAB workspace. If the Cancel push button is selected, the file dialog is closed and the tool returns to its previous state.

Notes

- The sampling frequency is automatically updated after the file is read. For *.au, *.voc, and *.wav, the sampling frequency is set to that stored in the file. For *.tim or flat integer files, the sampling frequency is set to $fs = 1$. The sampling frequency can be changed using the sampling frequency popup menu as described below.
- Loading signals into the tool directly from data files is more memory efficient and will hence, improve overall performance. The improved performance comes from the facts (1) creation of two copies of the signal is avoided (one in the workspace and one in the tool) and (2) creation of a possible third copy for the restore facility is eliminated since the signal can now be restored directly from the data file if the Workspace-Restore menu item is selected.

Workspace-Quit Menu Item

Purpose

Quit the Signal Model Tool.

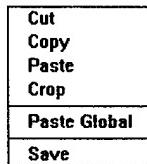
Use

1. Select the Quit menu item from the Workspace pull-down menu.

Results

The Signal Model Tool is closed. All edits are lost unless previously saved using the Workspace-Save menu item.

Edit Menu



Edit-Cut Menu Item

Purpose

Remove the section of the signal defined by the vertical cursors, placing the cut section into the cut-and-paste buffer.

Use

1. Define the section to be cut using the vertical cursors.
2. Select Cut from the Edit pull-down menu.

Result

The marked section is cut and the display is redrawn.

Notes

- The cut is inclusive (the data points directly under the cursors is included in the cut).
- The removed portion can be restored by immediately selecting Paste from the Edit pull-down menu.
- If the cursors are located at the beginning and end of the signal (i.e. the entire signal is marked for cutting), selecting the Cut push button will have no effect.
- The common signal is modified.

Edit-Copy Menu Item

Purpose

Copy the section of the signal defined by the vertical cursors and place the copied portion into the cut-and-paste buffer.

Use

1. Define the section to be copied using the vertical cursors.
2. Select Copy from the Edit pull-down menu.

Result

The copied section is placed into the cut-and-paste buffer.

Notes

- The copy is inclusive of the data under the cursors.

Edit-Paste Menu Item

Purpose

Insert the contents of the cut-and-paste buffer into the signal at a point selectable with

the mouse.

Use

1. Select Paste from the Edit pull-down menu.
2. The cursor changes into a crosshair and the display title changes to “Mark paste insertion point with cursor.”
3. Move the cursor to the desired insertion point and click the left mouse button.

Result

The contents of the cut-and-paste buffer are inserted into the signal and the signal display is redrawn.

Notes

- If the cut-and-paste buffer is empty, no action is performed (a cut or copy has to be performed first to fill the cut-and-paste buffer).
- The contents of the cut-and-paste buffer are not cleared. Multiple copies of the cut-and-paste buffer can be inserted by re-selecting the Paste push button.
- The pasted portion is simply included into the original signal. No transition smoothing is performed.
- The common signal is modified.

Edit-Crop Menu Item

Purpose

Delete portions of the signal that are outside the section delimited by the vertical cursors.

Use

1. Define the section to be retained using the vertical cursors.
2. Select Crop from the Edit pull-down menu.

Result

The signal outside the section delimited by the vertical cursors is deleted.

Notes

- The portion retained is inclusive of the data under the cursors.
- The deleted sections are not saved.
- The cut-and-paste buffer is not modified.
- The common signal is modified.

Edit-Paste Global Menu Item

Purpose

Insert the contents of the global cut-and-paste buffer into the signal at a point selectable with the mouse allowing cut-and-paste operations between signal edit tools.

Use

1. Select Paste Global from the Edit pull-down menu.

2. The cursor changes into a crosshair and the display title changes to "Mark paste insertion point with cursor."
3. Move the cursor to the desired insertion point and click the left mouse button.

Result

The contents of the global cut-and-paste buffer are inserted into the signal and the signal display is redrawn.

Notes

- The global cut-and-paste buffer content is that of the last cut or copy operation in any sigedit or sigmodel tool.
- All notes under Edit-Paste apply here.

Edit-Save Menu Item

Purpose

Save the section of the signal defined by the vertical cursors to a MATLAB workspace variable.

Use

1. Define the section to be saved using the vertical cursors.
2. Select Save from the Edit pull-down menu.
3. A dialog box will open in the tool window.
4. Enter the name to save the section under into the dialog edit box.
5. Press return or select the pushbutton labeled OK.

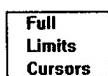
Results

The signal section is saved to the MATLAB workspace.

Notes

- If a variable with the same name already exists in the MATLAB workspace, it will be overwritten.

Play Menu



Play-Full Menu Item

Purpose

Send the signal to the workstation's audio output.

Use

1. Select Full from the Play pull-down menu.

Result

The entire signal is sent to the workstation's audio output.

Notes

- The signal is played at the sampling frequency set on the Sampling Freq popup menu on workstations supporting multiple output sampling frequencies.
- To play the modeled signal, save it to the MATLAB workspace and use the sound command.

Play-Limits Menu Item

Purpose

Send the signal displayed within the axis limits to the workstation's audio output.

Use

1. Use the zoom controls to zoom into the section of the signal to be played.
2. Select Limits from the Play pull-down menu.

Result

The signal displayed within the axis limits is sent to the workstation's audio output.

Notes

- The signal is played at the sampling frequency set on the Sampling Freq popup menu on workstations supporting multiple output sampling frequencies.

Play-Cursors Menu Item

Purpose

Send the signal displayed between the vertical cursors to the workstation's audio output.

Use

1. Define the section to be played using the vertical cursors.
2. Select Cursors from the Play pull-down menu.

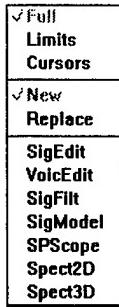
Result

The signal displayed between cursors is sent to the workstation's audio output.

Notes

- The signal is played at the sampling frequency set on the Sampling Freq popup menu on workstations supporting multiple output sampling frequencies.

Launch Menu



Purpose

Launch additional SPC Toolbox tools containing all or a portion of the current signal.

Use

1. Check the menu item corresponding to the signal portion to be used (Full, Limits or Cursors).
2. Check the menu item corresponding to launching a New tool or to Replace a previously opened tool of the same type (to be selected in step three).
3. Select the SPC Toolbox tool to launch.

Result

A new tool is launched containing the desired signal. The setting of the Sampling Freq popup menu is automatically passed to the new tool (if required).

Notes

- This option provides a quick and easy method for processing portions of a signal without having to go through a crop-save-load cycle.
- The number of additional tools that can be launched is limited only by the workstation capabilities (i.e. available memory).

Model-AR/MA/ARMA Menu Item



Purpose

Open the AR/MA/ARMA design tool (`armadsgn`) dialog box.

Use

1. Select AR/MA/ARMA from the Model pull-down menu.

Result

The `armadsgn` tool dialog box is opened. See `armadsgn` for more information on its use. Note that the model and period lengths are determined from the `sigmodel` tool setting rather than being the entire signal (as is the default when `armadsgn` is called directly from the command line).

Signal Model Tool Controls

Lock Starts Check Box

Purpose

Lock movement of model and period begin cursors (ME and PE edit boxes).

Use

1. Place the cursor on the Lock Starts check box and click the left mouse button until the check box is turned on or off as desired.

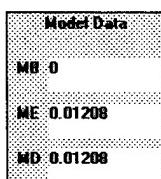
Result

If on, movement of either the model or period begin cursor using either the mouse or the ME or PE edit boxes will cause the other to move to the same location.

Notes

- If Lock Starts is turned on and the model and period begin cursors are not already at the same location, the period begin cursor will be moved to match the model begin cursor.

Model Data Edit Box Group



Purpose

Mark the beginning and end of the signal portion to use in generating the model.

Use

1. Enter the time index of the start of the data to be used in generating the model into the MB edit box.
2. Enter the time index of the end of the data to be used in generating the model into the ME edit box.
3. As an alternative to step two, a time offset to place the ME cursor from the MB cursor can be entered into the MD edit box.

Result

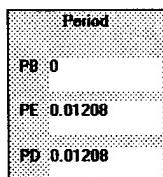
The data inclusive of the data points the Model Data cursors are attached to will be used in generating the desired model.

Notes

- If the sampling frequency is set to one, the time index will be given in sample number (first sample = one).
- As stated above, the data used in generating the model is inclusive of the data points the Model Data cursors are attached to. For example, if MB = 5, ME = 20, and MD = 15, the total length of data used in generating the model will be 16 data points.

- It is possible to move the Model Data cursors beyond the axis limits. It is also possible to move the cursors beyond the limits of the data. When the latter occurs, the Model Data cursors will be set to the first or last data sample in the signal as appropriate when the model is generated.

Period Edit Box Group



Purpose

Mark the beginning and end of the length of the model to be generated (model period).

Use

1. Enter the time index of the start of the model period into the PB edit box.
2. Enter the time index of the end of the model period into the PE edit box.
3. As an alternative to step two, a time offset to place the PE cursor from the PB cursor can be entered into the PD edit box.

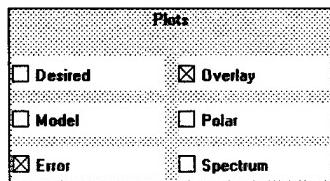
Result

The time inclusive of the data points the Period cursors are attached to will be used to determine the length of the model generated.

Notes

- If the sampling frequency is set to one, the time index will be given in sample number (first sample = one).
- As stated above, the length the model generated is inclusive of the data points the Period cursors are attached to. For example, if PB = 5, PE = 20, and PD = 15, the total length of the model generated will be 16 data points.
- It is possible to move the Period cursors beyond the axis limits. It is also possible to move the cursors beyond the limits of the data. This give rise to the possibility of generating models that are longer than the number of data points used to derive the model parameters.
- When a model length is requested that is longer than the number of data points between the Model Data start cursor and the end of the signal being model, the Desired signal will be zero padded. This will usually cause a jump in the error signal in the zero padded region. An example of this situation would occur with a 100 sample point signal when MB = 11, ME = 90, PB = 1, and PE = 100. Here, a model length of 100 has been requested but there are only 90 samples between MB and the end of the signal. Thus, the Desired signal would be zero padded with ten zeros when plotted and used to compute the error signal.

Plots Check Box Group



Purpose

Select the plots to be displayed.

Use

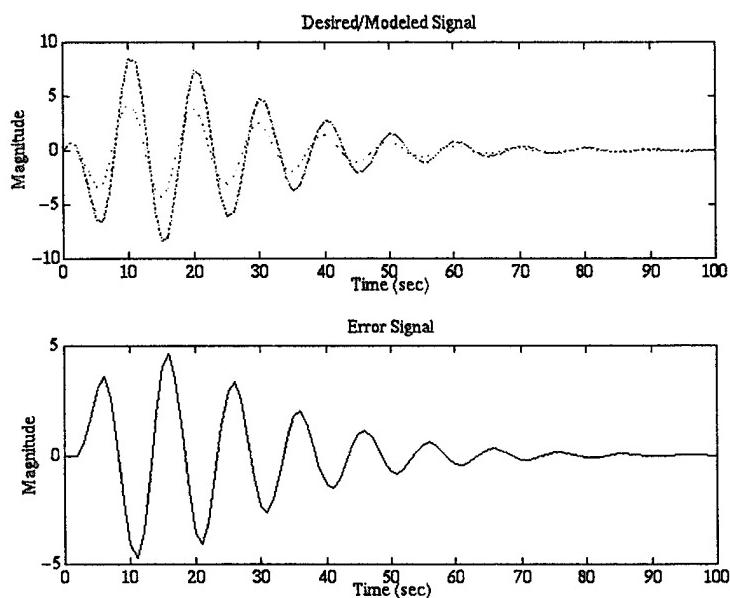
1. Place the cursor over the desired plot and click the left mouse button to turn the plot on or off.

Result

The desired plots are displayed when the model is generated.

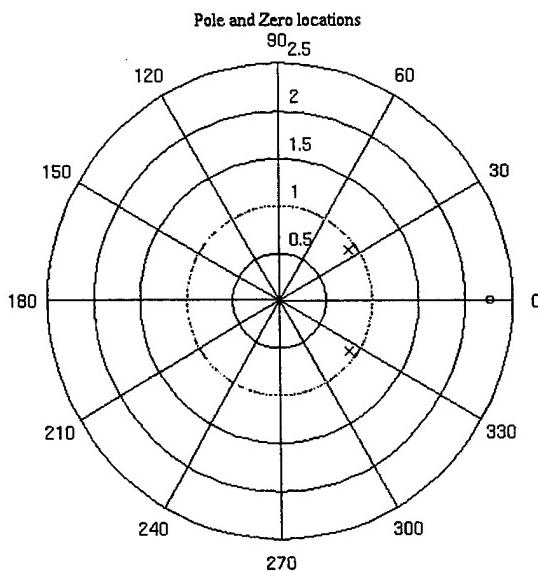
Notes

The plots are displayed in the following three figure windows:



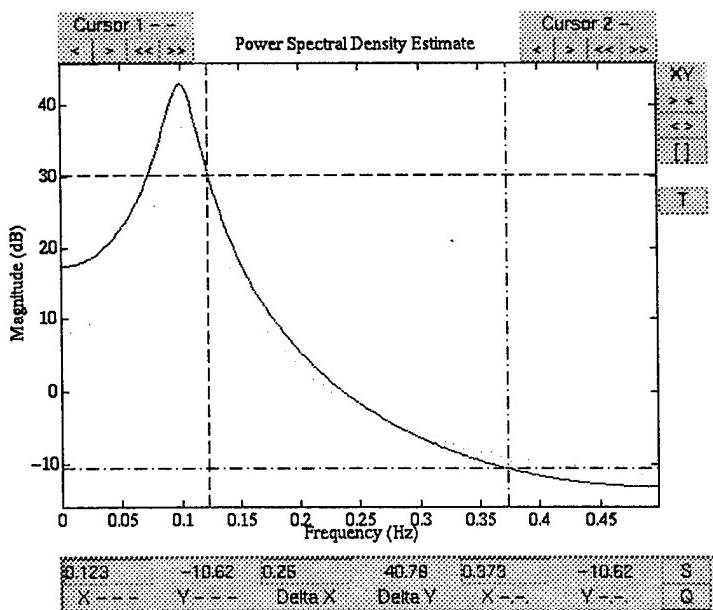
Model Window

- Desired - Data used in generating the model.
- Model - Modeled signal.
- Error - Error signal (desired minus model).
- Overlay - Overlay of model signal and desired signal.



Poles and Zeros Window

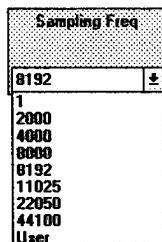
- Polar - Poles and zeros of model transfer function.



Spectral Estimate Window

- Spectrum - Spectral estimate based on model transfer function.

Sampling Freq Popup Menu



Purpose

Set the sampling frequency.

Use

1. Place the cursor on the Sampling Freq popup menu and click the left mouse button once to open the menu.
2. Select the sampling frequency with the left mouse button. If the sampling frequency is not available on the menu, the desired sampling frequency can be entered by selecting the User menu item. After User is selected, an edit box appears in place of the popup menu. Enter the desired sampling frequency into this edit box and press return.

Result

The display is redrawn with a new time axis according to the sampling frequency.

Notes

- If the sampling frequency is set to 1 Hz, the horizontal axis scale will correspond to the vector indices.

Examples

Load and model the speech phoneme /u/ contained in the audio signal stored in *uuu.mat*.

```
load uuu
sigmodel(uuu,1)
```

See the *sigmodel* tutorial section.

Related Files

- *sigmold.m* - Loads signal from workspace.
- *sigmosav.m* - Saves models to workspace.
- *sigmocal.m* - Callback functions for the Signal Model Tool.

See Also

armadsgn, *ar_corr*, *ar_covar*, *ar_mdcov*, *ar_burg*, *ar_prony*, *ar_shank*, *ar_durbn*, *zoomtool*

spect2d

Purpose

Frequency versus magnitude spectral estimation tool.

Synopsis

```
spect2d(x)
spect2d(x,fs)
spect2d('filename.ext')
spect2d('filename',bits)
```

Description

`spect2d(x)` opens a frequency versus magnitude spectral estimation tool for analyzing the time-domain signal x .

`spect2d(x,fs)` sets the sampling frequency to fs . The default sampling frequency is $fs = 8192$ Hz. A sampling frequency of $fs = 1$ Hz will display the frequency spectrum in digital frequency units.

`spect2d('filename.ext')` - Opens the file specified by '*filename.ext*' and loads the signal stored within the file directly into the tool. File formats with the extensions `*.au`, `*.voc`, `*.wav`, and `*.tim` can be specified by the '*filename.ext*' option only (be sure to include the extension). Pathnames are required when the file is not in the current directory. The sampling frequency is set to that stored in the file for `*.au`, `*.voc` and `*.wav` formats and to $fs = 1$ Hz for `*.tim` files. See the function `readsig` for more information on the functions used to read signal files.

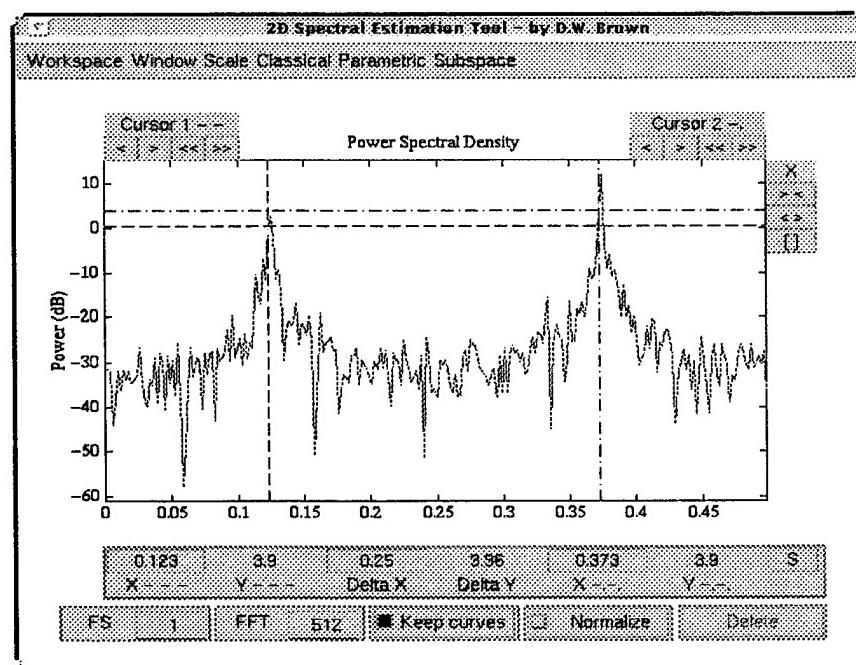
`spect2d('filename',bits)` - Opens the file specified by '*filename*' and loads the signal stored within directly into the tool as a flat integer formatted file. The number of bits-per-integer is specified by the *bits* argument where *bits* is either 8, 16, or 32. Integer formats must be stored as signed integers to be read correctly. Multi-byte integers (16 and 32) are assumed to be compatible with the format used by the workstation in use ("Little Endian" and "Big Endian" byte ordered files are not compatible). Pathnames are required when the file is not in the current directory. The sampling frequency is set according to the file type loaded. See the File-Load menu item description for more information.

For programmers, `spect2d` returns the handle to the figure window it was opened in. This is useful for keeping track of `spect2d` when it is called by a parent application. An example of this is when `spect2d` is launched from `sigedit`. See the `zoomprog.m` file for programming hints.

spect2d Spectral Estimation Tool Window

The `spect2d` Spectral Estimation Tool window opens displaying a Welch spectral estimate of the entire signal. This display can be used as a simple spectrum analyzer in its simplest use at this point. Cursors and digital readouts for measurement and zooming in on parts the spectrum are provided using `zoomtool`. As multiple spectral estimates are added to the spectral plot, the toggle feature of `zoomtool` can be used to toggle attachment of the cursors between the estimates. Information identifying the spectral estimate the cursors are currently attached to is shown as the x-axis label on the spectral plot. Along the bottom of the window are a number of controls which affect the

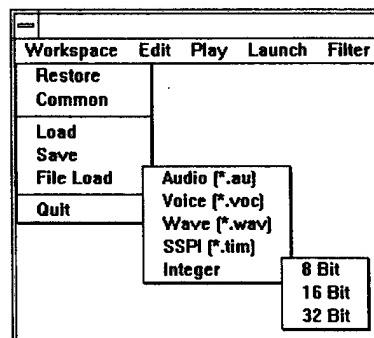
program's behavior. These will be discussed in more detail in the following sections.



Note: Only the positive frequencies are displayed. The DC component is removed before any spectral estimates are computed by subtracting the mean out of the time-domain signal and the DC frequency bin is forced to equal that of its closest positive neighbor.

spect2d Spectral Estimation Tool Pull-Down Menus

Workspace Menu



Workspace-Restore Menu Item

Purpose

Restore the original signal.

Use

1. Select the Restore menu item from the Workspace pull-down menu.

Results

The original signal is restored.

Notes

- Restore can be used to restore the original signal after a Workspace-Common or Workspace-Load operation.
- Restore saves a copy of the signal the tool was originally called with. Since saving a copy of long signal can be expensive in terms of memory, the user can turn off this ability. To turn the restore capability off, set the variable SPC_RESTORE to *off* in the configuration file spcolors.m. See the section on customizing SPC Tools for more information.
- The setting of the SPC_RESTORE variable is ignored when the filename option is used to start the tool. This is due to the fact that when the tool is started using a filename as the input argument, only the filename is saved in memory (not the signal) and thus, when Restore is selected, the data file can simply be read in again.

Workspace-Common Menu Item

Purpose

Load the global vector SPC_COMMON into the 2D spectral estimation tool. The SPC_COMMON vector is updated by all SPC tools that modify a signal (i.e. editing, filtering, etc.). The Common menu item therefore provides a means of passing signal between tools.

Use

1. Select the Common menu item from the Workspace pull-down menu.

Results

The common signal is loaded.

Workspace-Load Menu Item

Purpose

Load a signal from the MATLAB workspace.

Use

1. Select the Load menu item from the Workspace pull-down menu.
2. A dialog box will open in the tool window.
3. Enter the name of the variable to load into the edit box.
4. Press return or select the push button labeled OK.

Result

The new signal is loaded.

Notes

- If the variable does not exist in the MATLAB workspace, an error will occur.
- To abort loading a new signal, select the Cancel push button in the dialog box.

Workspace-File Load Menu Item

Purpose

Load a signal into the tool directly from a data file.

Use

1. Select the File Load menu item to open the file type submenu.
2. Select the file type to load. For Sun audio files, select the Audio (*.au) submenu item. For SoundBlaster Creative Voice files, select the Voice (*.voc) submenu item. For Microsoft Windows wave files, select the Wave (*.wav) submenu item. For Statistical Signal Processing Incorporated time-domain signal files, select the SSPI (*.tim) submenu item. For flat integer data files, select the Integer submenu item and then select the appropriate integer size from the Integer sub-submenu (8-Bits, 16-Bits or 32-Bits).
3. After selecting the appropriate file type, a file dialog box (produced by the MATLAB `uigetfile` command) will appear containing a list of available files. Navigate using the dialog box controls to the desired file and double-click on the filename or select the filename and click on the Done push button. To cancel loading a file, select the Cancel push button in the file dialog box

Results

If a file has been selected, it is read directly into the tool, bypassing the MATLAB workspace. If the Cancel push button is selected, the file dialog is closed and the tool returns to its previous state.

Notes

- The sampling frequency is automatically updated after the file is read. For *.au, *.voc, and *.wav, the sampling frequency is set to that stored in the file. For *.tim or flat integer files, the sampling frequency is set to $fs = 1$. The sampling frequency can be changed using the sampling frequency popup menu as described below.
- Loading signals into the tool directly from data files is more memory efficient and will hence, improve overall performance. The improved performance comes from the facts (1) creation of two copies of the signal is avoided (one in the workspace and one in the tool) and (2) creation of a possible third copy for the restore facility is eliminated since the signal can now be restored directly from the data file if the Workspace-Restore menu item is selected.

Workspace-Quit Menu Item

Purpose

Quit the *spect2d* tool.

Use

1. Select the Quit menu item from the Workspace pull-down menu.

Results

The *spect2d* tool is closed. All edits are lost unless previously saved using the Workspace-Save menu item.

Window Pull-Down Menu



Purpose

Select the data smoothing window used in smoothing classical, FFT based spectral estimates.

Use

1. Select the desired window from the Rectangular, Hamming, Hanning, Bartlett, or Blackman menu-items on the Window pull-down menu.

Result

The menu-item of the selected window is checked. The window will be applied to the next FFT based spectral estimate generated.

Notes

- The window is applied by computing a data smoothing window of the same length as the actual number data points used per frame. The window is then applied to the data and an FFT of the frame is computed. Any zero padding due to a frame length smaller than the FFT length takes place after the window has been applied. See the notes section under the Classical menu item descriptions for information on the frame length each classical method uses.

Scale Menu



Purpose

Select the scale of the frequency magnitude plot.

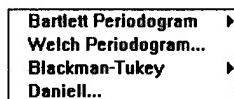
Use

1. Pull-down the Scale menu and select the desired Linear or Logarithmic scale menu item.

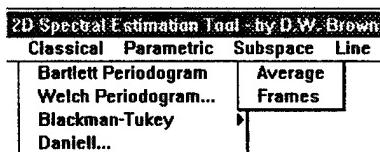
Result

The menu item of the selected scale is checked and the spectral plot redisplayed using the new scale.

Classical Menu



Classical-Bartlett Periodogram Menu Item



Purpose

Add an averaged periodogram (Bartlett procedure) spectral estimate to the spectral plot.

Use

1. Pull-down the Classical menu and select the Avg Periodogram menu item to open the sub-menu.
2. Select the Average submenu item to display the average of all frames in computing the spectral estimate or the Frames submenu item to display the spectrum of each individual frame.

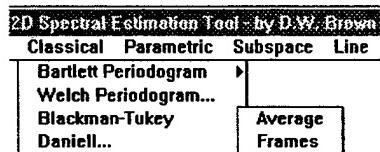
Result

An averaged periodogram spectral estimate is computed. If the Keep Curves check box is checked, the spectral estimate is added to the current spectral plot. If the Keep Curves check box is unchecked, all current spectral estimates in the spectral plot are deleted and replaced with the new spectral estimate.

Notes

- The Bartlett procedure uses non-overlapping frames.
- See the `avgpergm` command for additional information on how the spectral estimate is computed.
- When the cursors are attached to the spectral estimate curve for the average periodogram, the x-axis label of the spectral plot displays as `AvgFFT(nfft)`, where `nfft` is the size of the FFT used in computing the estimate, when the individual frames have been averaged. When attached to an individual frame, the x-axis is labeled in the form `AvgFFT(nfft,framenumbers)` where `framenumbers` is the frame number counted consecutively from the beginning of the signal.
- The FFT size corresponds to the frame size used in computing the spectral estimate.

Classical-Blackman Tukey Menu Item



Purpose

Add a correlogram (Blackman-Tukey procedure) spectral estimate to the spectral plot.

Use

1. Pull-down the Classical menu and select the Blackman-Tukey menu item.
2. Select the Average submenu item to display the average of all frames in computing the spectral estimate or the Frames submenu item to display the spectrum of each individual frame.

Result

A correlogram spectral estimate is computed. If the Keep Curves check box is checked, the spectral estimate is added to the current spectral plot. If the Keep Curves check box is unchecked, all current spectral estimates in the spectral plot are deleted and replaced with the new spectral estimate.

Notes

- See the `blacktuk` command for additional information on how the spectral estimate is computed.
- When the cursors are attached to the spectral estimate curve for the correlogram, the x-axis label of the spectral plot displays as *BlackTuk(N)*, where *N* is the number of samples-per-frame used in computing the estimate when the individual frames have been averaged. When attached to an individual frame, the x-axis is labeled in the form *BlackTuk(N,framenumbers)*, where *framenumbers* is the frame number counted consecutively from the beginning of the signal.
- The frame size used in computing the spectral estimate is the length of the FFT divided by two.

Classical-Daniell Menu Item

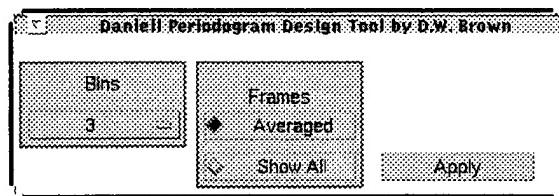
Purpose

Add a frequency smoothed, averaged periodogram (Daniell-Bartlett procedure) spectral estimate to the spectral plot.

Use

1. Pull-down the Classical menu and select the Daniell menu item. The following dialog

box will open.



2. In this dialog box, choose the number of adjacent frequency bins to average from the **Bins** popup menu. Then use the **Frames** radio button group to choose whether to display the average of the spectral estimate of all frames or display the spectral estimate of each individual frame. Once these two options have been set, select the **Apply** push button to generate the spectral estimate.

Result

A Daniell periodogram spectral estimate is computed. If the **Keep Curves** check box is checked, the spectral estimate is added to the current spectral plot. If the **Keep Curves** check box is unchecked, all current spectral estimates in the spectral plot are deleted and replaced with the new spectral estimate. The design dialog box is closed.

Notes

- See the **daniell** command for additional information on how the spectral estimate is computed.
- When the cursors are attached to the spectral estimate curve for the Daniell periodogram, the x-axis label of the spectral plot displays as *Daniell(nfft,bins)* where *nfft* is the FFT size used in computing the estimate when the individual frames have been averaged and *bins* is the number of adjacent frequency bins averaged. When attached to an individual frame, the x-axis is labeled in the form *Daniell(nfft,bins,framenumbers)*, where *framenumbers* is the frame number counted consecutively from the beginning of the signal.
- The frame length is the length of the FFT and non-overlapping frames are used in computing the spectral estimate.
- To abort, simply close the design dialog window using the platform windowing system.

Classical-Welch Menu Item

Purpose

Add an averaged periodogram (Welch procedure) spectral estimate to the spectral plot.

Use

1. Pull-down the Classical menu and select the Welch menu item.

Result

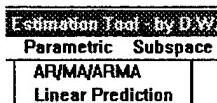
An averaged periodogram spectral estimate is computed. If the **Keep Curves** check box is checked, the spectral estimate is added to the current spectral plot. If the **Keep Curves** check box is unchecked, all current spectral estimates in the spectral plot are deleted

and replaced with the new spectral estimate.

Notes

- See the spectrum command in the MATLAB *Signal Processing Toolbox* for additional information on how the spectral estimate is computed.

Parametric Menu



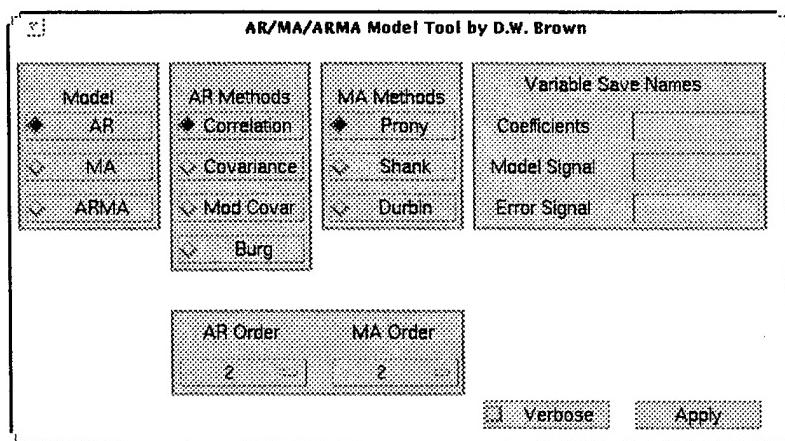
Parametric-AR/MA/ARMA Menu Item

Purpose

Add an auto-regressive (AR), moving-average (MA), or auto-regressive, moving-average (ARMA) model based spectral estimate to the spectral plot.

Use

1. Pull-down the Parametric menu and select the AR/MA/ARMA menu item. The following dialog box will open.



2. From this dialog box, use the Model, AR Methods, and MA Methods to select the desired modeling method. Use the AR Order and MA Order popup menus to set the desired model order. Once these options have been set, select the Apply push button to generate the spectral estimate. See the `armadsgn` command for more information on the AR/MA/ARMA Modeling Tool.

Result

An AR/MA/ARMA spectral estimate is computed. If the Keep Curves check box is checked, the spectral estimate is added to the current spectral plot. If the Keep Curves check box is unchecked, all current spectral estimates in the spectral plot are deleted and replaced with the new spectral estimate. The design dialog box is closed.

Notes

- See the `ar_corr`, `ar_covar`, `ar_mdcov`, `ar_burg`, `ar_prony`, `ar_shank`, and `ar_durbn` com-

- mands for additional information on how the spectral estimate is computed.
- When the cursors are attached to the spectral estimate curve for an AR/MA/ARMA spectral estimate, the x-axis label of the spectral plot displays as *AR(armodel, order)*, *MA(mamodel,maorder)*, or *ARMA(armodel,arorder,mamodel,maorder)*, where *armodel* and *mamodel* are the AR and MA models used and *arorder* and *maorder* are the model orders used in computing the spectral estimate.
 - To abort, simply close the design dialog window using the platform windowing system.

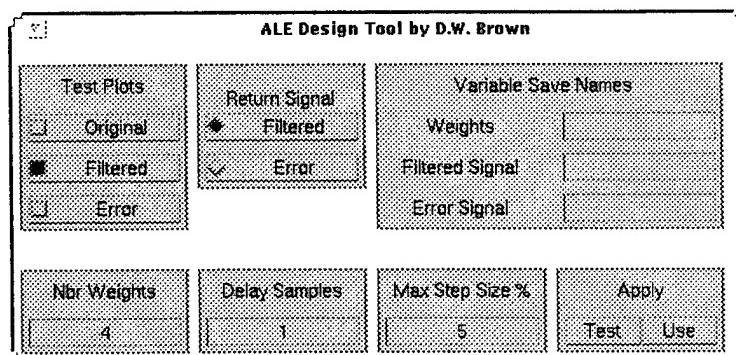
Parametric-Linear Prediction Menu Item

Purpose

Add a moving-average (MA) model based spectral estimate to the spectral plot using an adaptive, least-mean square linear prediction.

Use

1. Pull-down the Parametric menu and select the Linear Prediction menu item. The following dialog box will open.



2. From this dialog box, use the Nbr Weights, and Max Step Size edit boxes to select the desired adaptive LMS parameters. Once these options have been set, select the Use push button to generate the spectral estimate. See the `aledsgn` command for more information on the ALE Design Tool.

Result

A moving-average linear-prediction based spectral estimate is computed. If the Keep Curves check box is checked, the spectral estimate is added to the current spectral plot. If the Keep Curves check box is unchecked, all current spectral estimates in the spectral plot are deleted and replaced with the new spectral estimate. The design dialog box is closed.

Notes

- The Delay Samples edit box must be set to one for a one-step linear prediction estimate to result.
- See the `lmsale` command for additional information on how the spectral estimate is computed.
- When the cursors are attached to the spectral estimate curve for a linear prediction

spectral estimate, the x-axis label of the spectral plot displays as *LinPred(weights)*, where *weights* is the number of weights used in computing the spectral estimate.

Subspace Menu



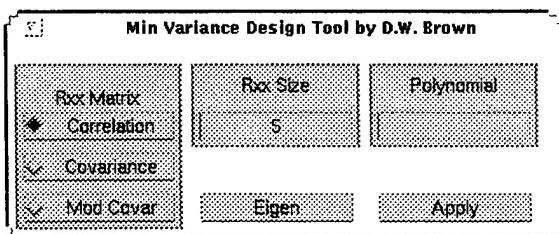
Subspace-Min Variance Menu Item

Purpose

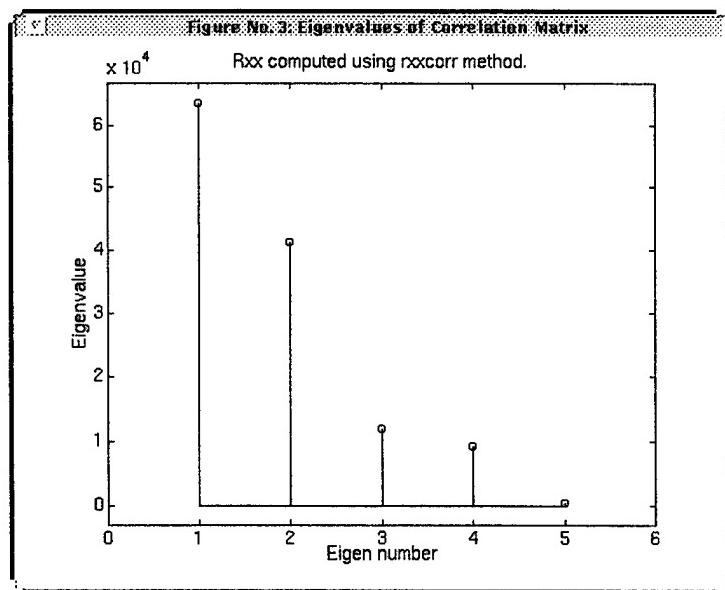
Add a minimum-variance (maximum-likelihood) based spectral estimate to the spectral plot.

Use

1. Pull-down the Subspace menu and select the Min Variance menu item. The following dialog box will open.



2. From this dialog box, use the Rxx Method radio button group and Rxx Size edit box to set the parameters of the correlation matrix estimate used in computing the spectral estimate. Once these two options have been set, select the Apply push button to generate the spectral estimate. In choosing the size of the correlation matrix to use, the Eigen push button can be used to display a plot of the eigenvalues of the correlation matrix.



Result

A minimum-variance based spectral estimate is computed. If the Keep Curves check box is checked, the spectral estimate is added to the current spectral plot. If the Keep Curves check box is unchecked, all current spectral estimates in the spectral plot are deleted and replaced with the new spectral estimate. The design dialog box and eigenvalue display window (if open) are closed.

Notes

- See the minvar command for additional information on how the spectral estimate is computed. See the ploteig command for the function used to plot the eigenvalues. See the rxxcorr, rxxcovar and rxmdcov for information about these methods of estimating the correlation matrix.
- When the cursors are attached to the spectral estimate curve for a minimum-variance spectral estimate, the x-axis label of the spectral plot displays as *MinVar(rxxmethod, size)*, where *rxxmethod* is the method used to estimate the correlation matrix and *size* is the size of the correlation matrix used in computing the spectral estimate.
- To abort, simply close the design dialog window using the platform windowing system.
- An equation form of the minimum variance spectral estimate can be obtained by entering a variable name into the Polynomial edit box before selecting the *Apply* push button. The spectral estimate will be in the form

$$S_x(e^{jk\omega}) = \frac{1}{\text{Polynomial}},$$

where the polynomial is defined by

$$a_{-(N-1)}e^{-j(N-1)\omega} + a_{-(N-2)}e^{-j(N-2)\omega} + \dots + a_{-1}e^{-j\omega} + 1 + a_1e^{j\omega} + \dots + a_{(N-1)}e^{j(N-1)\omega},$$

where N is the size of the correlation matrix.

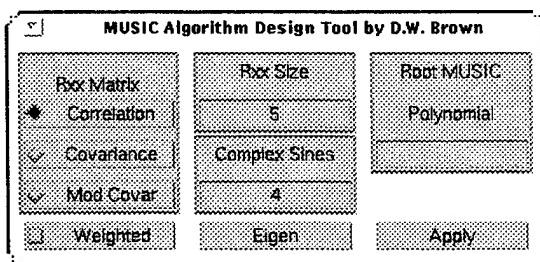
Subspace-MUSIC Menu Item

Purpose

Add a Multiple Signal Classification (MUSIC) based spectral estimate to the spectral plot.

Use

1. Pull-down the Subspace menu and select the MUSIC menu item. The following dialog box will open.



2. From this dialog box, use the Rxx Method radio button group and the Rxx Size edit box to select the desired parameters of the correlation matrix estimated used in the spectral estimate. Use the Complex Sines edit box to set the number of estimated complex sinusoidal signals located in the signal. Keep in mind that a real sinusoidal signal consists of two complex sinusoids. In choosing the size of the correlation matrix to use and the number of complex sines to enter, the Eigen push button can be used to display a plot of the eigenvalues of the correlation matrix. For an eigenvalue weighted MUSIC spectral estimate, check the Weighted check box. Once these options have been set, choose the Apply push button to generate the spectral estimate.

Result

A MUSIC based spectral estimate is computed. If the Keep Curves check box is checked, the spectral estimate is added to the current spectral plot. If the Keep Curves check box is unchecked, all current spectral estimates in the spectral plot are deleted and replaced with the new spectral estimate. The design dialog box and eigenvalue display window (if open) are closed.

Notes

- See the musicsp command for additional information on how the MUSIC spectral estimate is computed. See the musicspw command for additional information on how the weighted MUSIC spectral estimate is computed. See the poteig command for the

function used to plot the eigenvalues. See the `rxxcorr`, `rxxcovar` and `rxxmdcov` for information about these methods of estimating the correlation matrix.

- If the number entered in the Complex Sines edit box is one less than the size of the correlation matrix, a Pisarenko Harmonic Decomposition spectral estimate will result.
- Use of the `rxxmdcov` method to estimate the correlation matrix will result in what is commonly referred to as the Forward-Backward MUSIC spectral estimate.
- When the cursors are attached to the spectral estimate curve for a MUSIC spectral estimate, the x-axis label of the spectral plot displays as *MinVar(rxxmethod, size,sines)*, where *rxxmethod* is the method used to estimate the correlation matrix, *size* is the size of the correlation matrix used, and *sines* is the number of estimated complex sinusoids in the signal in computing the spectral estimate.
- To abort, simply close the design dialog window using the platform windowing system.
- Both the MUSIC and eigenvalue weighted MUSIC algorithms used are based on the noise subspace. The eigenvectors associated with the $N-K$ smallest eigenvalues are used to form the projection matrix, where N is the size of the correlation matrix, and K is the estimated number of complex sinusoids in the signal.
- An equation form of the MUSIC spectral estimate can be obtained by entering a variable name into the Root MUSIC Polynomial edit box before selecting the Apply push button. The spectral estimate will be in the form

$$S_x(e^{jk\omega}) = \frac{1}{\text{Root MUSIC Polynomial}},$$

where the root MUSIC polynomial is defined by

$$a_{-(N-1)}e^{-j(N-1)\omega} + a_{-(N-2)}e^{-j(N-2)\omega} + \dots + a_{-1}e^{-j\omega} + 1 + a_1e^{j\omega} + \dots + a_{(N-1)}e^{j(N-1)\omega},$$

where N is the size of the correlation matrix.

Lines Menu

Purpose

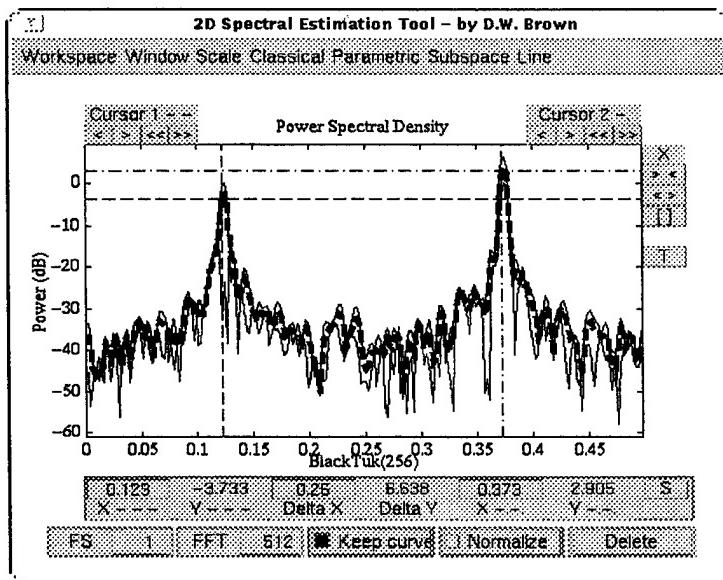
Customize the style, width, and color of the spectral estimate curve the cursors are currently attached to.

Use

1. Select the Lines menu.
2. Select the line property to modify (Style, Width, or Color).
3. Select the desired property value.

Result

The property of the spectral curve line the cursors are currently attached to is modified. The following figure shows how the Line menu was used to highlight the average when both the Classical-Blackman Tukey-Average and Classical-Blackman Tukey-Frames menu items were selected for spectral estimates.



Notes

- Lines-Width-Increase increases the line width by one unit. Lines-Width-Decrease decreases the line width by one unit. Depending upon the workstation monitor resolution, increasing or decreasing the line a width single step may or may not be noticeable.

spect2d Spectral Estimation Tool Controls

Delete Push Button

Purpose

Delete the spectral estimate the cursors are currently attached to.

Use

1. Toggle the cursor attachments until they are on the curve of the spectral estimate to delete.
2. Select the Delete push button.

Result

The curve the cursors are attached to is deleted and the cursors are attached to the next spectral estimate curve in the toggle order.

Notes

- The Delete push button is disabled when only one spectral estimate is present in the

spectral plot.

Sampling Frequency "FS" Popup Menu

Purpose

Set the sampling frequency.

Use

1. Place the cursor on the FS popup menu and click the left mouse button once to open the menu.
2. Place the cursor on the sampling frequency and click the left mouse button. If the sampling frequency is not available on the menu, the desired sampling frequency can be entered by selecting the User menu item. After User is selected, an edit box appears in place of the popup menu. Enter the desired sampling frequency into this edit box and press return.

Result

The display is redrawn with a new frequency axis according to the sampling frequency.

Notes

- In the popup menu, the User menu item is replaced by the sampling frequency entered using the edit box. To subsequently change the sampling frequency after previously using the edit box, the last sampling frequency on the popup menu has to be selected in place of the User menu item (this is where the User menu item was).

FFT Length Popup Menu

Purpose

Set the FFT length used in the FFT based spectral estimates and the number of points in the transfer function curve of parametric and subspace spectral estimates.

Use

1. Place the cursor on the FFT popup menu and click the left mouse button once to open the menu.
2. Place the cursor on the desired FFT length and click the left mouse button. If the desire FFT length is not available on the menu, it can be entered by selecting the User menu item. After User is selected, an edit box appears in place of the popup menu. Enter the desired FFT length into this edit box and press return.

Result

The new FFT length will be used in computing the next spectral estimate generated.

Notes

- In the popup menu, the User menu item is replaced by the FFT length entered using the edit box. To subsequently change the FFT length after previously using the edit box, the last FFT length on the popup menu has to be selected (this is where the User menu item was).

- Only the positive frequencies are displayed. Thus, out of a 512 point FFT, only 256 points are displayed.
- The frequency resolution between points is fs/FFT_length .

Keep curves Check Box

Purpose

Keep previous spectral estimates on the spectral plot.

Use

1. Place the mouse cursor over the check box and check or uncheck the box with the left mouse button.

Result

Subsequent spectral estimates are added to the spectral plot if checked. If not checked, subsequent spectral estimates replace previous estimates in the spectral plot.

Notes

- Unchecking the Keep Curves check box deletes previously saved curves.
- The Keep Curves check box is automatically unchecked whenever Workspace-Restore, Workspace-Common, Workspace-Load, Workspac-File Load, and Scale-Logarithmic menu items are selected, and whenever the Normalize check box or Load push button is selected.

Normalize Check Box

Purpose

Normalize the power in each spectral estimate such that the maximum peak in the estimate occurs at $S_x(f_{peak}) = 1 \text{ Watt} = 0 \text{ dB}$.

Use

1. Place the mouse cursor over the check box and check or uncheck the box with the left mouse button.

Result

The current spectral estimates and all subsequent spectral estimates are normalized if checked.

Notes

Once the spectral estimates have been normalized, they cannot be "un-normalized." A warning message to this effect is issued when unchecking this check box.

Examples

Load the speech signal contained in the audio file `seatsit.voc` into the tool.

```
s = loadvoc('seatsit');
spect2d(s)
```

or

spect2d

```
spect2d('seatsit.voc')
```

Algorithm

See the menu item description for the individual spectral estimation methods for more information.

Related Files

- `spect2dl.m` - Loads signal from workspace.
- `spectrm2.m` - Spectrum command modified to accept a window argument.

See Also

`spect3d`, `zoomtool`

spect3d

Purpose

Time versus frequency versus magnitude spectral surface tool.

Synopsis

```
spect3d(x)
spect3d(x,fs)
spect3d('filename.ext')
spect3d('filename',bits)
```

Description

`spect3d(x)` - Opens a time versus frequency versus magnitude spectrum tool for analyzing the time-domain signal *x*.

`spect3d(x,fs)` - Sets the sampling frequency to *fs*. The default sampling frequency is *fs* = 8192 Hz. A sampling frequency of *fs* = 1 Hz will display the frequency spectrum in normalized frequency units and the frame length in samples-per-frame vice time (see below).

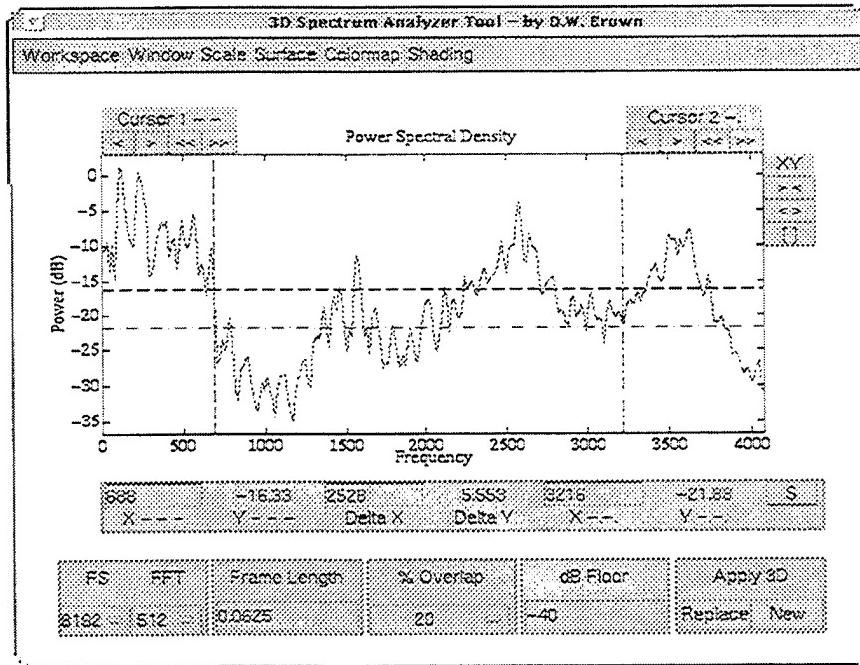
`spect3d('filename.ext')` - Opens the file specified by '*filename.ext*' and loads the signal stored within the file directly into the tool. File formats with the extensions **.au*, **.voc*, **.wav*, and **.tim* can be specified by the '*filename.ext*' option only (be sure to include the extension). Path names are required when the file is not in the current directory. The sampling frequency is set to that stored in the file for **.au*, **.voc* and **.wav* formats and to *fs* = 1 Hz for **.tim* files. See the function `readsig` for more information on the functions used to read signal files.

`spect3d('filename',bits)` - Opens the file specified by '*filename*' and loads the signal stored within directly into the tool as a flat integer formatted file. The number of bits-per-integer is specified by the *bits* argument where *bits* is either 8, 16, or 32. Integer formats must be stored as signed integers to be read correctly. Multi-byte integers (16 and 32) are assumed to be compatible with the format used by the workstation in use ("Little Endian" and "Big Endian" byte ordered files are not compatible). Path names are required when the file is not in the current directory. The sampling frequency is set according to the file type loaded. See the **File-Load** menu item description for more information.

For programmers, `spect3d` returns the handle to the figure window it was opened in. This is useful for keeping track of `spect3d` when is called by a parent application. An example of this is when `spect3d` is launched from `sigedit`. See the `zoomprog.m` file for programming hints.

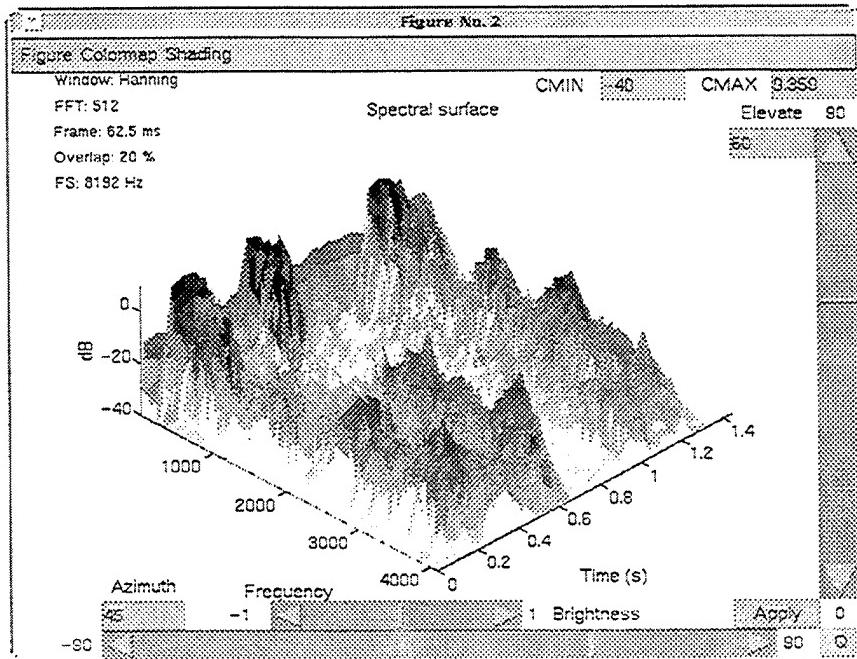
spect3d Spectrum Analyzer Tool Screen

The `spect3d` Spectrum Analyzer tool opens displaying a Welch spectral estimate of the entire signal. This display can be used as a spectrum analyzer. Cursors and digital readouts are available using `zoomtool` for measurement and zooming in on specific parts of the spectrum. Selecting of the **Apply 3D** push buttons will open a second window with a three-dimensional time/frequency/magnitude surface.



Frequency Band Selection

The spectrum's vertical cursors can be used to select a frequency band to display on the surface plots. Initially, the cursors are located at zero and $f_s/2$ so that the entire positive side of the spectrum is displayed. If desired, the user can move the cursors (using zoomtool cursor controls) to define a frequency band in between them. When a spectral surface is generated, only the frequency band defined by the cursors will appear on the three-dimensional plot.



Colormapping and Shading Selection

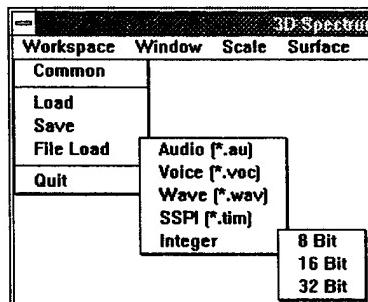
Default colormaps and shading can be selected before a spectral surface is generated. After generation, each surface figure window is opened with Colormap and Shading pull down menus. These menus allow the user to chose from a variety of colormaps and shading options. Selecting various color and shading schemes can be used to enhance the detail shown on the three-dimensional plot. Colormaps can also be inverted.

Legend

A legend is added to each surface figure detailing the FFT window, FFT length, frame length, frame overlap percentage and sampling frequency. For the pcolor option, the plot title is used in place of a legend to display this information.

spect3d Spectrum Tool Pull-Down Menus

Workspace Menu



Workspace-Common Menu Item

Purpose

Load the global vector SPC_COMMON into the spect3d spectrum tool. The SPC_COMMON vector is updated by all SPC tools that modify a signal (i.e. editing, filtering, etc.). The Common menu item therefore provides a means of passing signal between tools.

Use

1. Select the Common menu item from the Workspace pull-down menu.

Results

The common signal is loaded.

Workspace-Load Menu Item

Purpose

Load a signal from the MATLAB workspace.

Use

1. Select the Load menu item from the Workspace pull-down menu.

2. A dialog box will open in the tool window.
3. Enter the name of the variable to load into the edit box.
4. Press return or select the push button labeled OK.

Result

The new signal will be loaded.

Notes

- If the variable does not exist in the MATLAB workspace, an error will occur.
- To abort loading a new signal, select the Cancel push button in the dialog box.

Workspace-Save Menu Item

Purpose

Save the current spectral surface to the MATLAB workspace as a matrix.

Use

1. Select the Save menu item from the Workspace pull-down menu.
2. A dialog box will open in the tool window.
3. Enter the name to save the signal under into the dialog edit box.
4. Press return or select the push button labeled OK.

Results

The current signal is saved to the MATLAB workspace.

Notes

- Three variables are saved to the MATLAB workspace. The variable names will be in the form *name_id* where *name* is the variable name the user entered and *_id* identifies the variable contents as follows:
 - *name_ss* is a two-dimensional matrix containing the magnitude data
 - *name_f* is a vector containing the frequency scale.
 - *name_t* is a vector containing the time scale.
- If a variable with the same name exists in the MATLAB workspace, it will be overwritten.

Workspace-File Load Menu Item

Purpose

Load a signal into the tool directly from a data file.

Use

1. Select the File Load menu item to open the file type submenu.
2. Select the file type to load. For Sun audio files, select the Audio (*.au) submenu item. For SoundBlaster Creative Voice files, select the Voice (*.voc) submenu item. For Microsoft Windows wave files, select the Wave (*.wav) submenu item. For Statistical Signal Processing Incorporated time-domain signal files, select the SSPI (*.tim) submenu item. For flat integer data files, select the Integer submenu item and then select the appropriate integer size from the Integer sub-submenu (8-Bits, 16-Bits or

32-Bits).

3. After selecting the appropriate file type, a file dialog box (produced by the MATLAB `uigetfile` command) will appear containing a list of available files. Navigate using the dialog box controls to the desired file and double-click on the filename or select the filename and click on the Done push button. To cancel loading a file, select the Cancel push button in the file dialog box

Results

If a file has been selected, it is read directly into the tool, bypassing the MATLAB workspace. If the Cancel push button is selected, the file dialog is closed and the tool returns to its previous state.

Notes

- The sampling frequency is automatically updated after the file is read. For `*.au`, `*.voc`, and `*.wav`, the sampling frequency is set to that stored in the file. For `*.tim` or flat integer files, the sampling frequency is set to $f_s = 1$. The sampling frequency can be changed using the sampling frequency popup menu as described below.
- Loading signals into the tool directly from data files is more memory efficient and will hence, improve overall performance. The improved performance comes from the facts (1) creation of two copies of the signal is avoided (one in the workspace and one in the tool) and (2) creation of a possible third copy for the restore facility is eliminated since the signal can now be restored directly from the data file if the Workspace-Restore menu item is selected.

Workspace-Quit Menu Item

Purpose

Quit the spect3d tool.

Use

1. Select the Quit menu item from the Workspace pull-down menu.

Results

The spect3d tool is closed. All edits are lost unless previously saved using the Workspace-Save menu item.

Window Pull-Down Menu



Purpose

Select the window used in smoothing the FFT used in forming the spectral estimate.

Use

1. Select the desired window from the Rectangular, Triangular, Hamming, Hanning, Bartlett, or Blackman menu-items on the Window pull-down menu.

Result

The menu-item of the selected window is checked. The window is immediately applied to the Welch estimate contained in the tool. The window will be applied to the next spectral surface generated.

Notes

- The window is applied by computing a window containing the same number of actual data points per frame. The window is then applied to the data and the FFT is applied. Any zero padding takes place after the window has been applied.
- The triangular and Bartlett windows are identical when the window length is even. However, the triangular window is computed slightly differently than the Bartlett window when the window length is odd.

Scale Menu



Purpose

Select the scaling of the frequency magnitude plot (logarithmic or linear).

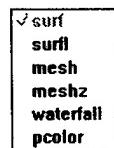
Use

1. Pull-down the Scale menu and select the desired Linear or Logarithmic menu item.

Result

The menu-item of the selected scale is checked. The selected scale is immediately applied to the Welch estimate contained in the tool. The window will be applied to the next spectral surface generated.

Surface Pull-Down Menu



Purpose

Select the 3D surface generated for the spectral estimate.

Use

1. Select the desired surface from the surf, surfl, mesh, meshz, waterfall, or pcolor menu-items in the Surface pull-down menu.

Result

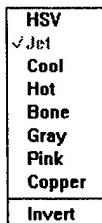
The menu-item of the selected surface is checked and will be used for the next surface generated.

Notes

- See the MATLAB Reference Guide form more information about each surface.

- The default surface is `surf`.
- The `pcolor` surface generates spectrogram type two-dimensional plots.

Colormap Pull-Down Menu



Purpose

Select the default colormap used for spectral surface.

Use

1. Select the desired surface from the `HSV`, `jet`, `cool`, `hot`, `Bone`, `gray`, or `copper` menu-items on the Colormap pull-down menu.
2. To invert the colormap, check the `Invert` menu item.

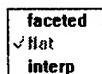
Result

The menu-item of the selected colormap is checked and will be used for the next surface generated.

Notes

- See the MATLAB Reference Guide for more information about each colormap.
- The default colormap is `hot`.

Shading Pull-Down Menu



Purpose

Select the default shading applied to the 3D surface.

Use

1. Select the desired shading method from the `faceted`, `flat`, or `interp` menu-items on the Shading pull-down menu.

Result

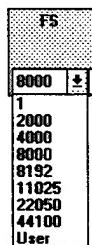
The menu-item of the selected shading method is checked and will be used for the next surface generated.

Notes

- See the MATLAB Reference Guide form more information about each shading method.
- The default shading method is `flat`.

spect3d Spectrum Tool Controls

Sampling Frequency "FS" Popup Menu



Purpose

Set the sampling frequency.

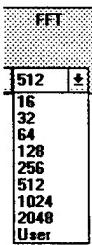
Use

1. Place the cursor on the FS popup menu and click the left mouse button once to open the menu.
2. Place the cursor on the sampling frequency and click the left mouse button. If the sampling frequency is not available on the menu, the desired sampling frequency can be entered by selecting the User menu item. After User is selected, an edit box appears in place of the popup menu. Enter the desired sampling frequency into this edit box and press return.

Result

The display is redrawn with a new frequency axis according to the sampling frequency.

FFT Length Popup Menu



Purpose

Set the FFT length used in the Welch and spectral surface estimates.

Use

1. Place the cursor on the FFT popup menu and click the left mouse button once to open the menu.
2. Place the cursor on the desired FFT length and click the left mouse button. If the desired FFT length is not available on the menu, it can be entered by selecting the User menu item. After User is selected, an edit box appears in place of the popup menu. Enter the desired FFT length into this edit box and press return.

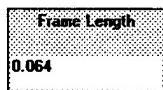
Result

The spectral estimate is recomputed and displayed. The new FFT length will be used in computing the next spectral surface generated.

Notes

- Only positive frequencies are displayed. Thus, out of a 512 point FFT, only 256 points are displayed.
- The frequency resolution between points is fs/FFT_length .

Frame Length Edit Box



Purpose

Set the length of the signal used in each frame.

Use

Time:

1. Enter the frame length in seconds. Any entry with a fractional portion (i.e. 1.5, 2.3 versus 2.0 or 2) is considered a time.

Samples-per-Frame:

1. Enter the frame length in samples-per-frame. Any integer can be used.

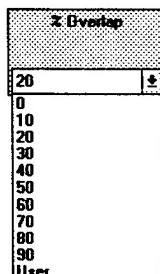
Result

The new frame length will be used in computing the next spectral estimate.

Notes

- The default value (that displayed when the tool opens) is given in seconds and equates to the FFT length number of samples at the specified sampling frequency.
- An error message will be displayed if the number of samples-per-frame is greater than the FFT length.

Frame Overlap Popup Menu



Purpose

Set the percentage of data used in computing successive frames overlap.

Use

1. Place the cursor on the % Overlap popup menu and click the left mouse button once to open the menu.
2. Place the cursor on the desired overlap percentage and click the left mouse button. If the desire overlap percentage is not available on the menu, it can be entered by selecting the User menu item. After User is selected, an edit box appears in place of the popup menu. Enter the desired overlap percentage into this edit box and press return.

Result

The new overlap percentage amount will be used in computing the next spectral surface generated.

Note: No overlap is used in computing the two-dimensional Welch spectral estimate.

dB Floor Edit Box



Purpose

Set the lowest dB magnitude to be display in spectral surface plots when using a logarithmic scale.

Use

1. Enter the value in decibels (this will usually be a negative number).

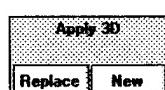
Result

Magnitudes below this minimum will be seen as a flat “floor” on spectral surface plots

Notes

- This control can be very useful for bringing out details in spectrogram (pcolor) plots.

Apply 3D Push Buttons



Purpose

Generate a new spectral surface plot.

Use

1. Select the New push button to generate a spectral surface in a new figure window.
2. Select the Replace push button to close the figure window which contains the last estimate before opening a new figure window to plot the new spectral surface.

Result

A new spectral estimate using the current parameters is generated and displayed in a new figure window.

Notes

- The new figure window will have the same size and screen location as the previous figure window if the previous figure window has been moved or resized and is still open when the New or Replace push button is selected.
- The v3dtool tool is added to all surface plots except pcolor. v3dtool allows the user to rotate the surface, change the colormap and shading and change the color axis after the surface has been generated.
- Generating several spectral surfaces may eventually lead to a possible memory bug on Sun workstations. This bug appears as the spectral surface partially rendering and then appearing to stop rendering without any error messages generated. Pressing control-C will return control of MATLAB to the user. It is assumed this is caused by memory fragmentation or leakage in version 4.1 operating on Sun workstations. The current recovery method is first to attempt to pack memory using the pack command. If this fails, save the workspace, quit, restart MATLAB, and then reload the workspace.

Examples

Load the speech signal contained in the audio file `seatsit.voc` and generate a spectral surface using the default parameters.

```
s = loadvoc('seatsit');
spect3d(s)
```

After the tool is opened, select the New push button.

Algorithm

The two-dimensional Welch spectral estimate is computed using the spectrum command from the MATLAB Signal Processing Toolbox. No overlap between successive frames is used. The function has been specially modified to use various windows.

The spectral surface is computed using a straight FFT. If the number of samples-per-frame is less than the FFT length, zero padding occurs. Before applying the FFT to a frame, the desired smoothing window is applied to the data used in the frame. The last frame may not be computed using the same number of samples-per-frame depending on the length of the data and frame used.

Related Files

- `spect3dl.m` - Loads signal from workspace.
- `spect2ds.m` - Saves spectral surface to workspace.
- `spectrm2.m` - Spectrum command modified to accept a window argument.

See Also

`spect2d`, `zoomtool`, `v3dtool`

v3dtool

Purpose

View 3D plot tool.

Synopsis

v3dtool

v3dtool(*figure*)

Description

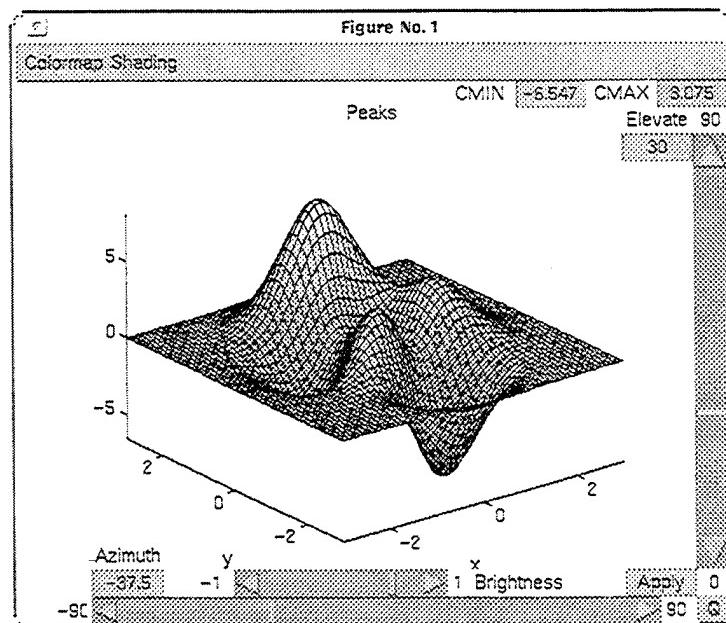
v3dtool(*figure*) starts the View 3D Tool in the figure window pointed to by the handle *figure* containing the three-dimensional plot. If the *figure* handle is not given, the current figure window is used. If the figure window does not have a three-dimensional plot or if there are more than one axis located within the figure window, error messages are produced and execution aborted.

v3dtool

v3dtool takes the drudgery out of typing the view command over and over to find the best orientation for three dimensional plots. It can also be used to easily change the color map, brightness and shading used to render the 3D plot. The most convenient use for v3dtool is in setting up 3D plots prior to printing. Once a plot has been setup, v3dtool can be removed leaving the plot as it was last modified using v3dtool.

v3dtool display

The View 3D Tool installs itself into the figure window containing the 3D plot by decreasing the size of the plot to make room for a number of edit box, push buttons



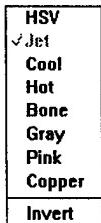
and slider controls which control the orientation and brightness of the 3D plot. A pull-down menu is also added with one menu containing various default colormaps and

another containing shading options. After v3dtool has been removed using the quit push button, the menus will be removed along with the controls and the size of the plot will return to that prior to executing v3dtool.

View 3D Tool Pull-Down Menus

The following pull-down menus are added to the figure window when v3dtool starts and are removed after the quit push button has been selected.

Colormap Pull-Down Menu



Purpose

Select the color map use to render the 3D plot.

Use

1. Select the desired surface from the hsv, jet, cool, hot, bone, gray, or copper menu-items on the Colormap pull-down menu.
2. To invert the color map, check the invert menu item.
3. Select the Apply push button with the mouse pointer.

Result

The menu-item of the selected colormap is checked and the 3d plot is redrawn using the new color map.

Notes

- See the MATLAB Reference Guide for more information about each color map.
- The default color map is hsv.

Shading Pull-Down Menu



Purpose

Select the shading applied to the 3D plot.

Use

1. Select the desired shading method from the faceted, flat, or interp menu-items on the Shading pull-down menu.
2. Select the Apply push button with the mouse pointer.

Result

The menu-item of the selected shading method is checked and the 3D plot is redrawn

using the new shading method.

Notes

- See the MATLAB Reference Guide form more information about each shading method.
- The default shading method is **faceted**.

View 3D Tool Controls

The following controls are added to the figure window when *v3dtool* starts and are removed after the quit push button has been selected.

Apply Push Button

Purpose

Apply changes made to the azimuth, elevation, color axis and shading method to the 3D plot.

Use

1. Set the desired plot attributes using the *v3dtool* controls and menus.
2. Select the **Apply** push button with the mouse pointer.

Result

The 3D plot is redrawn using the new settings.

Azimuth Edit Box and Slider

Purpose

Adjust the 3D plot azimuth (rotation about the z-axis).

Use

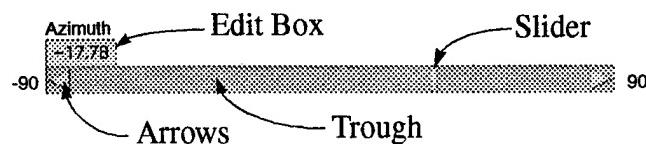
Edit Box

The Azimuth edit box is used to display the current azimuth of 3d plot. It it automatically updated to reflect the position of the slider position after the slider has been used to modify the plot azimuth. To explicitly change the azimuth to a specific angle:

1. Select the Azimuth edit box with the mouse pointer, enter the desired azimuth angle, and press return.

Slider

The Azimuth slider is used to modify the azimuth angle using the mouse pointer. The relative position of the slider control in the slider trough determines the azimuth angle which ranges from -90 degrees at the far left to 0 degrees at the center position to +90 degrees at the far right as shown below.



To change the azimuth angle using the slider:

1. Grab the slider with the mouse pointer and drag it to the location corresponding to the desired azimuth angle.

or

2. Click the mouse pointer in the slider trough on either side of the slider to move the slider toward the mouse pointer in approximately 18 degree jumps.

or

3. Click the mouse pointer on the slider arrows to change the azimuth angle in the direction of the arrow by approximately two degrees.

Result

If the edit box is used, the slider position is updated to reflect the value entered after the return key has been pressed. If the slider is used, the edit box is updated to reflect the new position of the slider. The 3D plot is not actually repositioned until the **Apply** push button is pressed.

Brightness Slider

Purpose

Increase or decrease the brightness of the color map.

Use

The Brightness slider is used to modify the brightness angle using the mouse pointer. The relative position of the slider control in the slider trough changes the brightness according to the **beta** value determined by the slider position. See the MATLAB **brighten** command for more information the **beta** value. The **beta** value ranges from -1 at the far left to 0 at the center position to +1 at the far right. To change the brightness using the slider:

1. Grab the slider with the mouse pointer and drag it to the location corresponding to the desired **beta** value.

or

2. Click the mouse pointer in the slider trough on either side of the slider to move the slider toward the mouse pointer in jumps of approximately two.

or

3. Click the mouse pointer on the slider arrows to change the beta value in the direction of the arrow in jumps of approximately 0.2.

Result

The brightness level is immediately adjusted to reflect the new beta value.

Color Axis "CMIN" and "CMAX" Edit Boxes

Purpose

Set the color axis limits. These can be used to position a particular region in the color map to the range of z-axis values in 3D plot or to "clip" the top or bottom of the 3D plot. See the MATLAB `caxis` command for more information.

Use

1. Enter the desired maximum and minimum values for the color axis scaling into the **CMAX** and **CMIN** edit boxes. The original values in these edit boxes will correspond to the highest and lowest z-axis values in the 3D plot.

Result

The color axis is adjusted after the **Apply** push button is selected.

Elevation Edit Box and Slider

Purpose

Adjust the 3D plot elevation (rotation above the plane from the x- and y-axis).

Use

Edit Box

The **Elevation** edit box is used to display the current elevation of 3D plot. It is automatically updated to reflect the position of the slider position after the slider has been used to modify the plot elevation. To explicitly change the elevation to a specific angle:

1. Select the **Elevation** edit box with the mouse pointer, enter the desired elevation angle (0 to 90 degrees), and press return.

Slider

The **Azimuth** slider is used to modify the azimuth angle using the mouse pointer. The relative position of the slider control in the slider trough determines the azimuth angle which ranges from 0 degrees at the bottom to 90 degrees at the top. To change the elevation angle using the slider:

1. Grab the slider with the mouse pointer and drag it to the location corresponding to the desired elevation angle.

or

2. Click the mouse pointer in the slider trough on either side of the slider to move the slider toward the mouse pointer in approximately 9 degree jumps.

or

3. Click the mouse pointer on the slider arrows to change the elevation angle in the direction of the arrow by approximately 1 degrees.

Result

If the edit box is used, the slider position is updated to reflect the value entered after the return key has been pressed. If the slider is used, the edit box is updated to reflect the new position of the slider. The 3D plot is not actually repositioned until the **Apply** push button is pressed.

"Q"uit Push Button

Purpose

Quit v3dtool, leaving the plot in its last condition.

Use

Select the "Q"uit push button after making all desired setting and rendering them by selecting the **Apply** push button.

Result

The v3dtool and its controls are removed and the axis returns to its original size.

Examples

Use v3dtool to change the orientation of the peaks display.

```
peaks;  
v3dtool
```

See Also

[zoomtool](#)

voicedit

Purpose

Visual speech signal editing tool.

Synopsis

```
voicedit(x);
voicedit(x,fs);
voicedit('filename.ext')
voicedit('filename',bits)
h = voicedit(x,fs)
```

Description

`voicedit(x)` opens a speech signal editing tool containing the time-domain signal `x`.

`voicedit(x,fs)` sets the sampling frequency to `fs`. The default sampling frequency is 8192 Hz. A sampling frequency of 1 Hz displays the time axis in sample number versus time.

`voicedit('filename.ext')` - Opens the file specified by '`filename.ext`' and loads the signal stored within the file directly into the tool. File formats with the extensions `*.au`, `*.voc`, `*.wav`, and `*.tim` can be specified by the '`filename.ext`' option only (be sure to include the extension). Path names are required when the file is not in the current directory. The sampling frequency is set to that stored in the file for `*.au`, `*.voc` and `*.wav` formats and to `fs = 1 Hz` for `*.tim` files. See the function `readsig` for more information on the functions used to read signal files.

`voicedit('filename',bits)` - Opens the file specified by '`filename`' and loads the signal stored within directly into the tool as a flat integer formatted file. The number of bits-per-integer is specified by the `bits` argument where `bits` is either 8, 16, or 32. Integer formats must be stored as signed integers to be read correctly. Multi-byte integers (16 and 32) are assumed to be compatible with the format used by the workstation in use ("Little Endian" and "Big Endian" byte ordered files are not compatible). The directory path to the file is required when the file is not in the current directory. The sampling frequency is set according to the file type loaded. See the `File-Load` menu item description for more information.

`h = voicedit(x,fs)` returns the handle to the figure window it was opened in. This is useful for programmers in keeping track of `sigedit` when it is called by a parent application. Such an example occurs when `voicedit` is launched from `sigedit`. See `zoomprog.m` for programming hints.

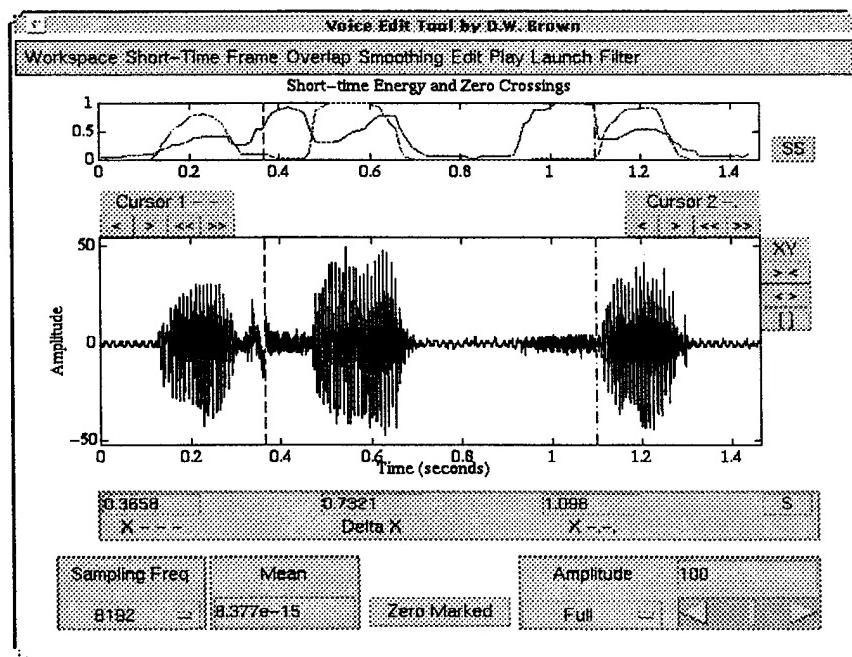
Note: No output arguments are supported. Edited signals must be saved using the `Save` menu item under the `Workspace` pull-down menu.

Voice Edit Tool

The Voice Edit Tool provides an interactive, graphical environment within MATLAB to edit speech signals represented as vectors. With the Voice Edit Tool, you have all the functionality of the Signal Edit Tool (`sigedit`) with the enhanced capability of displaying the short-time energy, magnitude and zero-crossing curves as aids in distinguishing between voiced and unvoiced phonemes and periods of silence.

Voice Edit Tool Screen

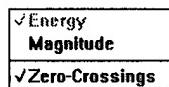
The Voice Edit Tool screen consists of a set of pull-down menus and a group of controls along with an axis containing the signal to be edited and a smaller axes containing the short-time information curves. The time-domain axis uses zoomtool to provide zoom capabilities along with cursor controls and cursor position readouts. Only the vertical cursors of zoomtool are displayed. The short-time information curve axes also contains vertical cursors whose positions match the time-domain axes cursors. Cursors in either axes can be “grabbed” and moved to new locations. Once the move is complete, the corresponding cursor in the other axis is moved to match. This feature allows the user to position the cursors in the short-time information axes directly on phonetic transitions indicated by the short-time information curves. Using zoomtool to zoom the time-domain axes also causes the short-time information axes to be correspondingly zoomed.



Voice Edit Tool Pull-Down Menus

In addition to all pull-down menus and controls contained in the signal edit tool (sigedit), the Voice Edit Tool contains the following pull-down menus for controlling the display of the short-time information curves.

Short-Time Menu



Short-Time - Energy Menu Item

Purpose

When checked, displays the short-time energy curve.

Use

1. Select the Energy menu item from the Traces pull-down menu.

Results

The short-time magnitude curve is deleted and the short-time energy curve is displayed.

Notes

- Raised portions of the short-time energy curve generally correspond to voiced phonemes.

Short-Time - Magnitude Menu Item

Purpose

When checked, displays the short-time magnitude curve.

Use

1. Select the Magnitude menu item from the Traces pull-down menu.

Results

The short-time energy curve is deleted and the short-time magnitude curve is displayed.

Notes

- Raised portions of the short-time magnitude curve generally correspond to voiced phonemes.

Short-Time - Zero Crossings Menu Item

Purpose

When checked, displays the short-time zero-crossing curve.

Use

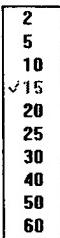
1. Select the Zero Crossings menu item from the Traces pull-down menu.

Results

The display of the short-time zero-crossing curve is either turned on or off dependent upon whether the Zero Crossings menu item is checked or not.

Notes

- Raised portions of the short-time zero-crossing curve generally correspond to unvoiced phonemes.

Frame Pull-Down Menu**Purpose**

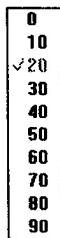
Selects the frame length used in computing the short-time information curves.

Use

1. Select the menu item containing the desire frame length (shown in milliseconds) from the Frame pull-down menu.

Results

- The short-time curves are recomputed and displayed.

Overlap Pull-Down Menu**Purpose**

Selects the overlap between successive frames used in computing the short-time information curves.

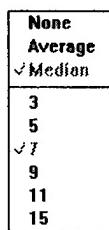
Use

1. Select the menu item containing the desired frame overlap (shown in percentage) from the Overlap pull-down menu.

Results

- The short-time curves are recomputed and displayed.

Smoothing Menu



Smoothing Pull-Down Menu - Method Menu Items

Purpose

Select the smoothing applied to the short-time information curves.

Use

1. Select one of the None, Average or Median menu items from the Smoothing pull-down menu.

Results

The short-time curves are recomputed and displayed.

Notes

- The median smoothing method generally provides the best distinction between voiced and unvoiced phonemes.

Smoothing Pull-Down Menu - Filter Length Menu Items

Purpose

Select the number of neighboring samples used in smoothing the short-time information curves.

Use

1. Select the menu item containing the desired number of samples from the Smoothing pull-down menu.

Results

The short-time curves are recomputed and displayed.

Notes

- The smoothing performed is non-causal. For example, if a smoothing length of three is selected, sample n is smoothed using samples $n-1$, n , and $n+1$.

Examples

Load the speech signal contained in the audio file `seatsit.voc`.

```
s = loadvoc('seatsit');
voicedit(s)
```

Related Files

- `voiceload.m` - Loads signal from workspace.
- `voicesave.m` - Saves edited signal to workspace.
- `voiccall.m` - Callback function library.

See Also

`sigedit`, `zoomtool`, `sp_steng`, `sp_stmag`, `sp_stzcr`

zoomtool

Purpose

2D graph zoom/measurement tool.

Synopsis

```
zoomtool  
zoomtool(h)  
zoomtool(h,'PropertyName','PropertyValue',...)
```

Description

`zoomtool` attaches `zoomtool` to the current axes object in the current figure window.

`zoomtool(h)` attaches `zoomtool` to the axis pointed to by the axes object handle `h`.

`zoomtool(h,'PropertyName','PropertyValue',....)` attaches `zoomtool` to the axis pointed to by the axes object handle `h` with the specified properties set. The properties allow `zoomtool` to be used in special situations and also as part of a larger Graphical User Interface (GUI) application.

Note: Only one `zoomtool` can be active in a single figure window at a time. Multiple `zoomtools` can be active as long as they are attached to axes in different figure windows.

zoomtool

`zoomtool` is useful for displaying and inspecting vector data such as digitized signals. The tool allows the user to select the exact portion of a vector to be studied. It also allows the user to directly read data point values, making it quite useful for measuring peak values and similar features.

`zoomtool` can be executed in any figure window containing a two-dimensional plot with one or more lines. The lines contained in the plot are restricted to those which have their X-axis values in sequentially increasing order with a constant interval between successive values. For example, the line created with

```
>> plot([-2 1 4 7 10],[2 8 3 9 1])
```

is allowed because the X-axis data increases sequentially by three. The line created by

```
>> plot([2 8 3 9 1],[-2 1 4 7 10])
```

is not allowed because the X-axis values are not sequential. The line

```
>> plot([-2 8 3 9 1])
```

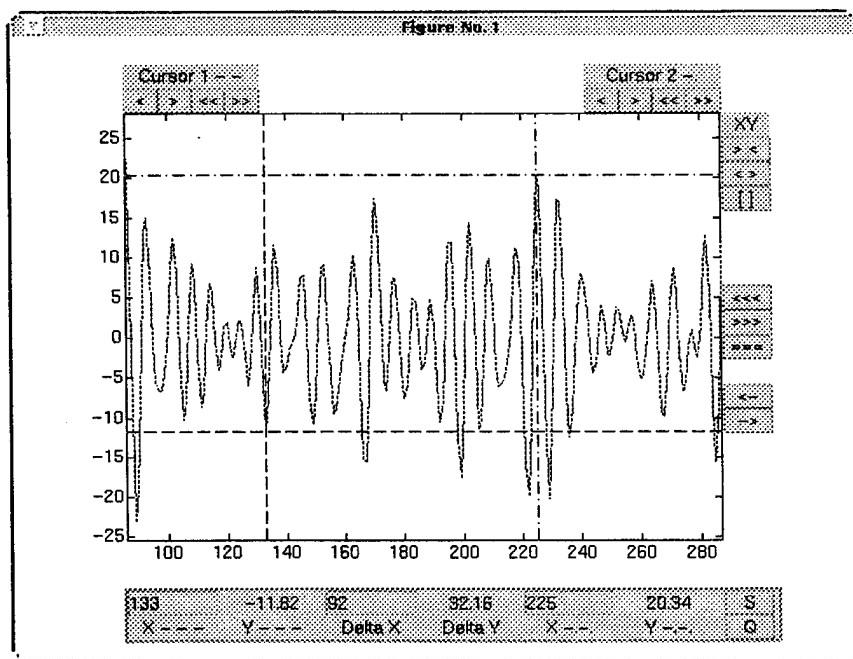
is allowed since MATLAB defaults the X-axis values to the vector indices [1 2 3 4 5]. A second restriction is that all lines must have the same X-axis endpoints if more than one line is contained in the plot.

Note: `zoomtool` will not work properly with the following types of plots because these do not meet the above conditions: bode plots, stem plots, scatter plots, any plot created with grid lines,

Zoomtool Display

zoomtool gets installed into a figure by decreasing the size of the axis making room for several push button controls, edit boxes and digital readouts. The tool tries to be as unobtrusive as possible (i.e. not interfere with any other plots or objects contained in a figure window) but may not be able to if the axis or figure window is too small. In these cases, try increasing the size of the figure window.

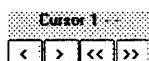
zoomtool draws two sets of cursor control buttons on top of the axis and draws zoom control, toggle, and scroll control push buttons down the right side of the axis. The toggle button is not drawn if the axis only contains a single line. The scroll controls are not drawn unless the axis has been zoomed. Along the bottom of the axis, zoomtool draws a readout bar containing the X- and Y-cursor positions and a readout for the difference between the two cursors. The X-axis readouts are edit boxes which allow the user to position the horizontal cursors by entering an absolute position (see below). The snapshot and quit push buttons are also drawn on the readout bar. The user can quit zoomtool by selecting the quit push button, at which time all controls and readouts will be removed and the axis returned to its original size.



Zoomtool Controls

Horizontal Cursor Control Groups

The cursor push button controls (one set for each cursor) allow movement of the vertical cursor along the X-axis. The horizontal cursor is automatically adjusted to the magnitude of the corresponding sample at the vertical cursor location. The push button controls are:



- "<" and ">" move the vertical cursor to the next left or right sample.
- "<<" and ">>" move the vertical cursor to the next left or right peak.

A peak is defined as the next maxima or minima in the curve.

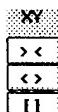
In addition to cursor manipulation with push buttons, the cursors can also be manipulated with the mouse. A single click on a curve will move the nearest vertical cursor to that point. The mouse button can also be held down and the nearest vertical cursor "dragged" to the desired location. Two modes of dragging the vertical cursor are supported. The first mode is by pressing the mouse button while the cursor is on the zoomed line. In this case, the nearest vertical cursor is "grabbed" and during the drag, the corresponding horizontal cursor is dragged along too. The second mode is by grabbing the vertical cursor by pressing the mouse button with the mouse cursor over the vertical cursor. Note that dragging the vertical cursor grabbed in this manner does not drag along the corresponding horizontal cursor. Once the mouse button is released, the horizontal cursor is moved. This second method is much smoother than the first one when the plot contains a large number of tightly spaced data points.

Note: Cursors are tied to actual data points. After a cursor has been moved, it is attached to the nearest data point.

Zoom Control Group

The zoom push button controls are drawn down the right side of the axis. There are actually three sets of zoom push buttons, one set for the zooming the X-axis and Y-axis individually and one for zooming both the X- and Y-axis together. The default operation is the display of the combined X- and Y-axis zoom push buttons only. The individual axis push button groups can be displayed by setting the appropriate property value.

The zoom push button controls are:



- "> <" zooms in on the X-axis between the vertical cursors.
- "< >" zooms out to the previously zoomed limits.
- "[]" zooms out to the full axis limits.

Zooming only takes place along the X-axis. Y-axis zooming refers to adjusting the Y-axis to maximize the view of the data between the vertical cursors.

Cursor Readout Group

0	-18.77	3984	19.69	3984	-30.46	S
X ...	Y ...	Delta X	Delta Y	X ...	Y ...	

The cursor readout group is drawn below the axis. Three pairs of readouts are displayed, one for each cursor pair and one for the absolute difference between the two. Normally, all readouts are displayed however, the Y-axis readouts (horizontal cursor positions) are not displayed if the horizontal cursors have been turned off by setting

the appropriate property.

The X-axis readouts (vertical cursor readouts) are edit boxes. Using these edit boxes, a desired location for the vertical cursors can be entered. After pressing return, the cursor is moved to the new location. The "Delta X" readout is also an edit box. It can be used to enter an offset to place cursor 2 from cursor 1. After pressing return, cursor 2 is moved to the desired offset. Entering a location beyond the limits of the axis moves the affected cursor to the axis limit. Note that cursors are attached to actual data points and so values for locations entered are rounded so that the cursor will lie on the closest data point.

"T"oggle Push Button



The toggle push button "T" toggles the attachment of the cursor to the next curve when an axis contains more than one curve. Toggling to a specific curve can be accomplished with the mouse by simply selecting the desire line. If there is only a single line in the plot, the toggle push button will not be displayed. The ability to toggle between multiple lines can be turned off by setting the **Toggle** property off.

Scroll Push Button Group

The scroll push button controls allow the user to scroll to the left or right in increments of twenty percent of the current axis range. Since scrolling can only take place after the display has been zoomed, the scroll push buttons are not displayed when the axis is zoomed to its full limits. Additionally, if the display is at the left or right limits of the data, only the scroll push button in the available scroll direction is displayed.

The scroll push buttons are:



- "<<<" scrolls the vector to the left (forward in time).
- ">>>" scrolls the vector to the right (backwards in time).
- "===" stops the scroll.
- "<->" scrolls the vector left twenty percent of the current limits.
- "<->" scrolls the vector right twenty percent of the current limits.

Scrolling automatically stops when the limits of the data has been reached. On slow processors or video displays, scrolling may seem "jerky" but is unavoidable due to the hardware limitations. Under MS-Windows MATLAB, the cursor will change to an hourglass while scrolling is taking place. Panning can still be stopped by clicking the "===" push button with the center of the hourglass over the push button.

"S"napshot Push Button

The snapshot push button allows the user to create a copy of the zoomed axis in a new figure window. Selecting snapshot opens a new window and recreates the plot in its currently zoomed condition. zoomtool then forgets about this window, allowing the user to treat it like any other graphics window. The user can change its title, labels, add text, and print this figure like any other MATLAB figure window.

"Q"uit Push Button

Quitting zoomtool with the quit push button ("Q") leaves the axis in the last zoomed state, removing the button bars and readouts.

Zoomtool Properties

The following properties can be set when zoomtool is started. Unlike MATLAB Handle Graphics Objects, the use of the `set` and `get` commands are not supported for these properties. The full property name must be used in specifying properties.

CursorCallBack string

Control Action. This property specifies any legal MATLAB expression, including the name of any M-file or function to be called after a zoomtool cursor has been moved. This property is useful to the programmer when zoomtool is included as part of a larger GUI applications interface.

HorizontalCursors on | off

Display horizontal cursors. This property determines whether or not the horizontal cursor and associated Y-axis readouts are displayed.

Scroll on | off

Allow scrolling. This property determines whether or not the user is allowed to scroll the display.

QuitPushButton on | off

Display "Q"uit push button. This property should be set to off when zoomtool is included as part of a larger GUI applications.

SnapShotPushButton on | off

Display "S"napshot push button. This property determines whether or not the user is allowed to make snapshots of the zoomed axis.

Toggle on | off

Allow toggling between multiple lines. This property determines whether the user can toggle between lines when multiple lines are contained in the plot. It also determines whether the "T"oggle push button is displayed or not.

ZoomX on | off

Display X-axis zoom controls. This property determines whether or not the independent X-axis zoom control push buttons are displayed.

ZoomY on | off

Display Y-axis zoom controls. This property determines whether or not the independent Y-axis zoom control push buttons are displayed.

ZoomXY on | off

Display X-axis zoom controls. This property determines whether or not the combined X- and Y-axis zoom control push buttons are displayed.

ZoomCallBack string

Control Action. This property specifies any legal MATLAB expression, including the name of any M-file or function to be called after zoomtool a zoom operation. This property is useful to the programmer when zoomtool is included as part of a larger GUI application interface.

Examples

Use zoomtool as an oscilloscope to measure the period of a 100 Hz sinusoid.

```
fs = 1000; % sampling frequency
ind = 0:1/fs:0.1; % time index
y = sin(2 * pi * 100 .* ind); % generate sinusoid
plot(ind,y); % plot sine wave
zoomtool(gca); % start zoomtool
```

- Use the cursor controls to move the cursor to the maxima of two successive waveforms and read the period from the Delta X readout.

Use zoomtool to measure the null-to-null bandwidth of a band pass filter.

```
[b,a] = cheby2(10,5,[.4 .5]); % design filter
[h,w] = freqz(b,a,512); % compute transfer function
mag = 20*log10(abs(h)); % compute magnitude
w = w/pi; % convert to scale digital frequencies
plot(w,mag); % plot transfer function
zoomtool(gca); % start zoomtool
```

- Use the cursor controls to move the cursors to either side of the passband and read the bandwidth from the Delta X readout.

Use zoomtool to find the index of the impulse function resulting from the correlation of gaussian white noise.

```
s = randn(256,1) * 2; % a "noise" signal
t1 = randn(1024,1); % a noise source
t2 = randn(1024,1); % a second noise source
t1(1:256) = t1(1:256) + s; % add signal to first noise
t2(201:200+256) = t2(201:200+256) + s; % offset signal into second noise
t1t2xx = xcorr(t1,t2); % compute cross-correlation
plot(t1t2xx); % plot the cross-correlation
zoomtool % start zoomtool
```

- Use the edit box to position cursor 1 at sample 1024 (identically zero in the cross-correlation). Grab cursor 2 and drag it into close proximity to the impulse. Use the

zoomtool

cursor 2 peak ("<<" or ">>") to place the cursor exactly on the impulse. The index of the impulse can be read from the cursor 2 X-axis readout. Note the Delta X readout displays the offset of the "signal."

Related Files

Any file beginning with "zoom."

See Also

Commands beginning with "zoom."

zoomcust

Purpose

Add line customization menu to zoomtool

Synopsis

```
zoomcust
zoomcust(figure)
```

Description

zoomcust(figure) adds a menu allowing the user to customize the style, width and color of the line the cursors are currently attached to in *zoomtool* active in the figure window pointed to by the handle *figure*. If *zoomtool* is not already active in the figure window, it is executed automatically.

The customization menu is

Line	
Style	
Solid	
Dash	
Dot	
Dash-Dot	
Width	
Increase	
Decrease	
Color	
Black	
Blue	
Cyan	
Green	
Magenta	
Red	
White	
Yellow	

The line width is increased and decreased by one unit. This may not always be visible depending upon the monitor resolution but should be visible when printed.

Example

You've created a plot of a line and used *zoomtool* to zoom in a portion of interest. Before printing the plot however, you would like to change the line style. Add a line customization menu to *zoomtool* to do this.

```
plot(randn(100,1))
zoomtool
zoomcust
```

See Also

zoomtool

zoomplay

Purpose

Add a menu to zoomtool to play portion of vector on the audio output.

Synopsis

zoomtool
zoomplay(*figure*)
zoomplay(*figure,fs*)

Description

zoomplay(figure) adds a Play menu to the figure window containing the figure pointed to by the handle *figure*. This menu allows the user to send all or a portion of the vector the zoomtool cursors are currently attached to. If the handle *figure* is not given, the current figure is used. If zoomtool is not already active in the figure window, it is executed automatically.

zoomplay(figure,fs) sets the default sampling frequency used to send the vector to the audio output. If the figure window contains a popup menu labeled 'Sampling Freq' or 'FS', the value currently displayed on the popup menu will override the default. If no default sampling frequency is given, a value of $fs = 8192 \text{ Hz}$ is used.

The Play menu is

- Play
 - Full
 - Limits
 - Cursors
- Full - plays the entire vector.
Limits - plays portion displayed between axis limits.
Cursors - play portion displayed between cursors.

As a programming tool:

zoomplay(figure,'portion') sends the specified portion of the vector the zoomtool cursors are attached to. The 'portion' can be either 'full', 'limits', or 'cursors'.

zoomplay(figure,'portion',fs) overrides the frequency display on a popup menu labeled 'Sampling Freq' or 'FS' if present. The default sampling frequency is $fs = 8192 \text{ Hz}$.

Example

Load and plot the signal contained in the file *seatsit.voc*. Plot the signal and add zoomtool to inspect the signal. After inspecting the signal, you decide you would like to hear a portion of it played over the workstation audio output. Use zoomplay to add this capability.

```
seatsit = loadvoc('seatsit.voc');
plot(seatsit)
zoomtool
zoomplay(gcf,8000)
```

See Also

zoomtool

zoomprog

Purpose

Add a menu to zoomtool to launch SPC Toolbox GUI tools.

Synopsis

```
zoomprog
zoomprog(figure)
```

Description

*zoomprog(*figure*)* adds a Launch menu to the figure window pointed to by the handle *figure*. The Launch menu allows the user to launch SPC Toolbox GUI tools with all or a portion of the vector the zoomtool cursors are currently attached to. If the handle *figure* is not given, the current figure is used. If zoomtool is not already active in the figure window, it is executed automatically.

The Launch menu is

Launch	
Full	- Sends the entire vector.
Limits	- Sends portion between axis limits.
Cursors	- Sends portion between cursors.

New	- Launches a new tool.
Replace	- Replaces previous tool (same type).

SigEdit	- Signal Editor
VoiceEdit	- Voice Signal Editor
SigFilt	- Signal Filtering Tool
SigModel	- Signal Modeling
SPScope	- Spectrum Scope (Analyzer)
Spect2D	- 2D Spectral Estimation
Spect3D	- 3D Spectral Analysis

Use

1. Check the menu item corresponding to the signal portion to be used (Full, Limits or Cursors).
2. Check the menu item corresponding to launching a New tool or to Replace a previously opened tool of the same type (to be selected in step three).
3. Select the SPC Toolbox tool to launch.

Example

You know that you want to start a spectrum tool with a portion of a signal but you do not know exactly where. Use zoomtool and zoomprog to find that portion and launch a spectrum tool.

```
seatsit = loadvoc('seatsit')
plot(seatsit)
zoomprog
```

See Also

[zoomtool](#)

zoomprog

Communication Function Reference

This section contains detailed descriptions of all communication functions in the Signal Processing and Communications Toolbox. It begins with a list of functions grouped by subject area and continues with the reference entries in alphabetical order. Information is also available through the online help facility.

Baseband Signal Generation

<code>antipodal</code>	Antipodal [-1,1] signal
<code>unipolar</code>	Unipolar [0,1] signal
<code>coswave</code>	Cosine waveform
<code>sinwave</code>	Sine waveform
<code>sawwave</code>	Sawtooth waveform
<code>sqwave</code>	Square waveform
<code>triwave</code>	Triangular waveform

Passband Signal Generation

<code>ook</code>	On-off keyed signal
<code>ookmsg</code>	On-off keyed signal with message
<code>bfsk</code>	Binary-frequency shift keyed signal
<code>bfskmsg</code>	Binary-frequency shift keyed signal with message
<code>bpsk</code>	Binary-phase shift keyed signal
<code>bpskmsg</code>	Binary-phase shift keyed signal with message

Modulation

<code>dsbsc</code>	Double-sideband, suppressed-carrier amplitude modulation
<code>dsblc</code>	Double-sideband, large-carrier amplitude modulation

Modulation

modindex	AM modulation index
envelope	AM envelope detection/demodulation
proddmod	Product demodulation by lowpass filtering and bit-rate subsampling
corrdmod	Product demodulation by bit synchronized integration

Signal Coding

lrs	Maximal-length, linear-recursive sequence
str2masc	String-to-binary-modem ASCII conversion
masc2str	Binary-modem-ASCII to string conversion

Signal Mixing

setsnr	Set total power ratio
setsnrbw	Set bandwidth power ratio

antipodal

Purpose

Generate a baseband, antipodal [-1,1] signal.

Synopsis

```
y = antipodal(rb,msg)
y = antipodal(rb,nbrbits)
y = antipodal(rb,msg,fs)
y = antipodal(rb,nbrbits,fs)
```

Description

`antipodal(rb,msg)` - Generates a baseband anti-podal signal at bit rate, rb , with the bit pattern specified in msg . The input argument msg must be a vector containing 1's and 0's. The sampling frequency defaults to $fs = 8192$ Hz.

`antipodal(rb,nbrbits)` - Generates a baseband anti-podal signal at bit rate, rb , with a random bit pattern $nbrbits$ long. The message is generated as a random sequence of 1's and -1's with $\text{Pr}(1)=\text{Pr}(-1)=0.5$. The sampling frequency defaults to $fs = 8192$ Hz.

The length of the output vector, y , can be computed by the formula:

$$\text{length} = nbrbaud \cdot \text{floor}\left(\frac{fs}{rb}\right)$$

`antipodal(rb,msg,fs)` and `antipodal(rb,nbrbits,fs)` - Sampling frequency is set to fs .

Note: If the bit rate, rb , and the sampling frequency, fs , are both equal to one, the resulting vector will contain alternating values of [-1,1].

Example

Generate 30 bits of a 50 bit-per-second signal sampled at 8000 Hz. Use this baseband signal as the modulating signal input to `dsbsc` to generate phase-shift keyed signal.

```
x = antipodal(50,30,8000);
y = dsbsc(x,2000,8000);
```

Limitations

- Due to the psuedo-random number generator, the probability of a one occurring 50% of the time will not be exact for short messages but will approach 50% as the message length increases.
- If the sampling frequency is not an exact multiple of the bit rate, unexpected results may occur. For example: `antipodal(9,18,20)` will produce an output of 36 samples vice the expected 40 samples (i.e. $18 / 9 = 2$ seconds at 20 samples-per-second). This occurs since $\text{floor}(fs/rb) = 2$ samples-per-bit.

See Also

`unipolar`, `sqwave`

bfsk, bfskmsg

Purpose

Generate binary frequency-shift keyed, bandpass signals.

Synopsis

```
y = bfsk(Rb,shift,fc);  
y = bfsk(Rb,shift,fc,duration);  
y = bfsk(Rb,shift,fc,duration,fs);  
y = bfskmsg(Rb,shift,fc,msg);  
y = bfskmsg(Rb,shift,fc,fs,msg);
```

Description

`bfsk(Rb,shift,fc)` - Generates one second of a binary frequency-shift keyed signal with a bit rate of Rb bits-per-second and a frequency shift of $shift$ centered at a carrier frequency, fc . Default sampling rate is 8192 Hz.

`bfsk(Rb,shift,fc,duration)` - Generates $duration$ seconds of the binary frequency-shift keyed signal.

`bfsk(Rb,shift,fc,duration,fs)` - Set the sampling frequency to fs .

`bfskmsg(Rb,shift,fc,msg)` or `bfskmsg(Rb,shift,fc,fs,msg)` - Generates a bandpass binary frequency-shift keyed signal with a bit rate of Rb bits-per-second, a frequency shift, $shift$, centered at a carrier frequency, fc , sampled at sampling frequency, fs , and containing the message, msg . If fs is not given, fs defaults to 8192 Hz.

Example

Generate a 64 bit-per-second, binary frequency-shift keyed signal at a carrier frequency of 2048 Hz and a frequency shift of 128 Hz lasting one-and-one-half seconds. Play the signal over the workstation's audio output.

```
y = bfsk(64,128,2048,1.5);  
sound(y,8192);
```

Algorithm

The binary frequency-shift keyed signal is generated by first creating two on-off keyed signals at the desired bit/sampling rates and combining. The on-off keyed signals are generated such that the mark signal is generated at $fc+shift/2$ and the space signal is generated at $fc-shift/2$. After the signals are generated, they are added to form a BFSK signal.

See Also

`unipolar`, `dsbsc`, `ook`, `ookmsg`, `bpsk`, `bpskmsg`

Reference

[1] Leon W. Couch, *Digital and Analog Communication Systems*, pp. 332-344, 351-356, Macmillan, 1990.

bpsk, bpskmsg

Purpose

Generate binary phase-shift keyed, bandpass signals.

Synopsis

```
y = bpsk(Rb,fc);
y = bpsk(Rb,fc,duration);
y = bpsk(Rb,fc,duration,fs);
y = bpsk(Rb,fc,msg);
y = bpskmsg(Rb,fc,fs,msg);
```

Description

bpsk(Rb,fc) - Generates one second of a binary phase-shift keyed signal with a bit rate of *Rb* bits-per-second centered at a carrier frequency, *fc*. Default sampling rate is 8192 Hz.

bpsk(Rb,fc,duration) - Generates *duration* seconds of a phase-shift keyed signal.

bpsk(Rb,fc,duration,fs) - Sets the sampling frequency to *fs*.

bpskmsg(Rb,fc,msg) or **bpskmsg(Rb,fc,fs,msg)** - Generates a binary phase-shift keyed signal with a bit rate of *Rb* bits-per-second, at a carrier frequency, *fc*, sampled at sampling frequency, *fs*, and containing the message, *msg*. If *fs* is not given, *fs* defaults to 8192 Hz.

Example

Generate a 64 bit-per-second, binary phase-shift keyed signal at a carrier frequency of 2048 Hz lasting one-and-one-half seconds. Play the signal over the workstation's speaker.

```
y = bpsk(64,2048,1.5);
sound(y,8192);
```

Algorithm

The binary phase-shift keyed signal is generated by first creating a binary antipodal signal at the desired bit/sampling rates and then applying a double-sideband, suppressed-carrier modulation scheme to the resulting baseband signal at the desired carrier frequency.

See Also

bpskmsg, ook, ookmsg, bfsk, bfskmsg

Reference

[1] Leon W. Couch, *Digital and Analog Communication Systems*, pp. 332-336, Macmillan, 1990.

corrdmod

Purpose

Demodulate a bandpass digital signal using a correlation demodulator.

Synopsis

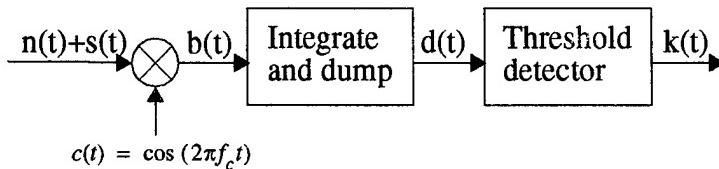
`y = corrdmod(x,Rb,fc)`

`y = corrdmod(x,Rb,fc,fs)`

Description

`corrdmod(x,Rb,fc)` - Demodulates the bandpass digital signal x using a correlation demodulator [1]. A correlation demodulator is defined here as a sinusoidal mixer followed by an integrate-and-dump filter and a threshold detector. The integration period is over a single bit-period defined by $T = 1/Rb$. A cosine with frequency fc Hz with zero-phase shift is used as the input to the sinusoidal mixer. This detector is a coherent detector, and thus requires the carrier used in modulating the bandpass digital signal to be a cosine wave with zero-phase shift also. The return variable y is a vector containing the demodulated bit-stream. The default sampling frequency is $fs=8192$ Hz.

`corrdmod(x,Rb,fc,fs)` - Sets the sampling frequency to f_s .



Example

Generate a BPSK signal containing the binary message [1 0 1 0 0 1] and demodulate using a correlation demodulator.

```

>> y = bpskmsg(64,1024,[1 0 1 0 0 1]);
>> corrdmod(y,64,1024)
ans =
    1
    0
    1
    0
    0
    1
  
```

Algorithm

The input signal, $n(t)+s(t)$, is multiplied by $\cos(2\pi f_c t)$, reshaped to form a matrix where each column contains the samples for one bit period, then each column is summed to perform the integration. Next, the integration output, $d(t)$, is passed

through a threshold detector with the threshold set so that

$$k(t) = \begin{cases} 1 & d(t) \geq 0 \\ 0 & d(t) < 0 \end{cases}$$

Limitations

The carrier frequency must be a multiple of the bit rate (same number of samples per bit).

See Also

`proddmod`, `bpsk`, `bpskmsg`, `bfsk`, `bfskmsg`, `ook`, `ookmsg`

Reference

[1] John G. Proakis, *Digital Communications*, pp 234-241, McGraw Hill, 1989.

coswave, sinwave

Purpose

Generate a baseband, cosine or sine wave.

Synopsis

```
y = coswave(fb);  
y = coswave(fb,duration);  
y = coswave(fb,duration,fs);  
y = coswave(fb,duration,fs,phaseshift);  
y = sinwave(fb);  
y = sinwave(fb,duration);  
y = sinwave(fb,duration,fs);  
y = sinwave(fb,duration,fs,phaseshift);
```

Description

coswave(fb) or *sinwave(fb)* - Generates one second of a sinusoidal wave at cycle frequency, *fb*, and a sampling frequency of 8192 Hz.

coswave(fb,duration) or *sinwave(fb,duration)* - Generates *duration* seconds of a sinusoidal wave at cycle frequency, *fb*, and a sampling frequency of 8192 Hz.

coswave(fb,duration,fs) or *sinwave(fb,duration,fs)* - Generates *duration* seconds of a sinusoidal wave at cycle frequency, *fb*, and sampling frequency, *fs*.

coswave(fb,duration,fs,phaseshift) or *sinwave(fb,duration,fs,phaseshift)* - Generates *duration* seconds of a sinusoidal wave at cycle frequency, *fb*, and sampling frequency, *fs*, with phase shift, *phaseshift*.

Examples

Use a 10 Hz sinusoidal modulating signal to generate an AM signal with a modulation index of 0.5 and a carrier frequency of 50 Hz, sampled at 128 Hz.

```
y = coswave(10,0.5,128);  
yy = dsblc(y,0.5, 50,128);
```

Generate 3425 samples of a 800 cycle-per-second sinusoidal wave sampled at 7000 Hz.

```
y = coswave(800,3425/7000,7000)
```

See Also

sqwave, *trrowave*, *sawwave*

dsblc

Purpose

Generate a double-sideband, large-carrier amplitude modulated signal.

Synopsis

```
y = dsblc(modsig,m,fc,fs)
```

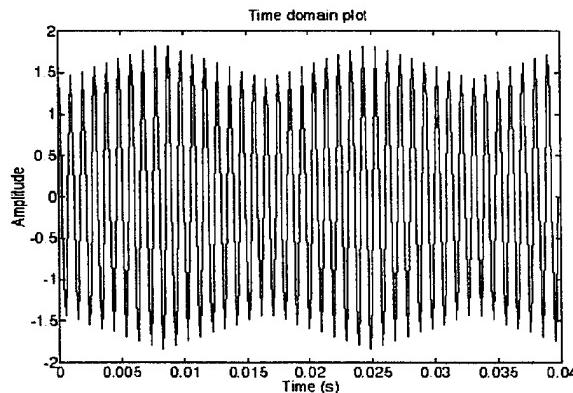
Description

`dsblc(modsig,m,fc,fs)` - Generates a double-sideband, large-carrier amplitude modulated signal, *y*. The signal contained in the vector *modsig* is modulated onto a sinusoidal carrier of amplitude 1 at carrier frequency, *fc*. The maximum peak-to-peak deviation of the modulating signal must vary evenly about zero [i.e. $(\max(\text{modsig}) == \text{abs}(\min(\text{modsig}))]$. *modsig* is multiplied by the modulation index, *m*, before being modulated onto the carrier. The length of the signal returned is equal to the length of *modsig*. The carrier is generated as a cosine with zero phase shift. The sampling frequency, *fs*, must be the same sampling frequency as that used to generate the modulating signal.

Example

Modulate one second of a 60 cycle triangular wave using a carrier of 1024 Hz with a modulation index of 0.3.

```
x = triwave(60,30);
y = dsblc(x,0.3,1024);
plottime(y,0.04);
```



Algorithm

The standard equation for double-sideband, suppressed-carrier amplitude modulation with zero phase shift.

$$y(t) = (1 + m f(t)) \cos(2\pi f_c t)$$

See Also

`dsbsc`, `modindex`, `envelope`

dsbsc**Purpose**

Generate a double-sideband, suppressed-carrier amplitude modulated signal.

Synopsis

```
y = dsbsc(modsig,fc,fs)
```

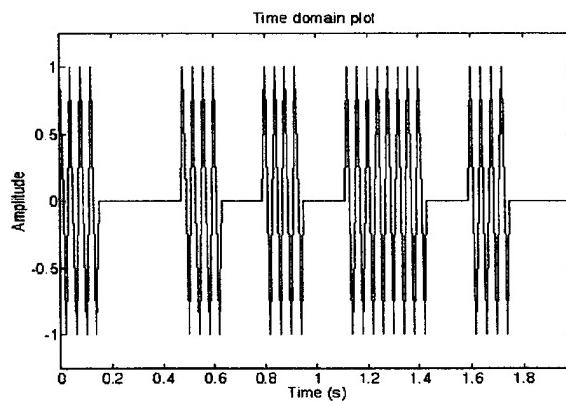
Description

`dsbsc(modsig,fc,fs)` - Generates a double-sideband, suppressed-carrier amplitude modulated signal, *y*. The signal contained in the vector *modsig* is modulated onto a sinusoidal carrier of amplitude 1 with a carrier frequency *fc*. The length of the modulated signal returned is equal to the length of the *modsig* vector. The carrier is generated as a cosine wave with zero phase shift. The sampling frequency, *fs*, must be the same sampling frequency used to generate the modulating signal.

Example

Generate 16 bits of an 8 bit-per-second signal sampled at 128 Hz. Use this baseband signal as the modulating signal input to `dsbsc` to generate an on-off keyed signal.

```
x = unipolar(8,[1 0 0 1 0 1 0 1 1 0 0 1 1 0 1],128);
y = dsbsc(x,32,128);
```

**Algorithm**

The standard equation for double-sideband, suppressed-carrier amplitude modulation with zero phase shift is given by:

$$y(t) = m(t) \cos(2\pi f_c t)$$

See Also

`dsbsc`

envelope

Purpose

Demodulate an amplitude modulated signal using envelope (noncoherent) detection.

Synopsis

```
[y,m] = envelope(x);
```

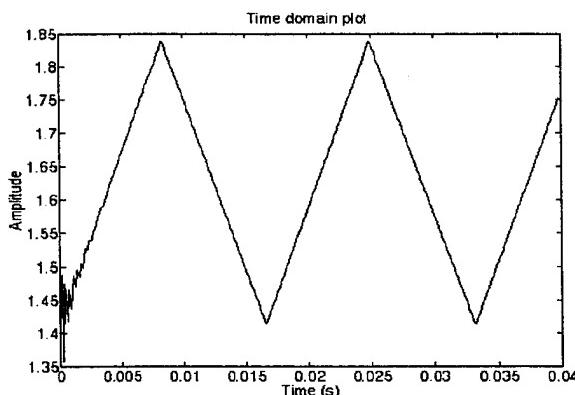
Description

`envelope(X)` - Returns the AM envelope of the input signal, x , and the modulation index, m .

Example

Demodulate using envelope detection the `dsblc` modulated triangular wave used in the `dsblc` command example.

```
x = triwave(60,30);
y = dsblc(x,0.3,1024);
z = envelope(y);
plottime(z,0.04);
```



Note: Transients are due to the FIR implementation of the Hilbert Transform.

Algorithm

The complex envelope of the signal x is obtained by computing the Hilbert transform of x . The resulting real envelope is the magnitude of the complex envelope.

See Also

`dsbsc`, `dsblc`, `modindex`

lrs**Purpose**

Generate a linear recursive sequence.

Synopsis

```
y = lrs(r,n);
y = lrs(r,n,fill,taps);
```

Description

`lrs(r,n)` - Returns n bits of a binary, maximal-length, linear recursive sequence using an r bit shift register. Uses a randomly generated fill with randomly generated taps.

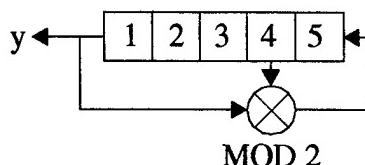
`lrs(r,n,fill,taps)` - Sets the initial fill and tap locations to those specified in the vectors *fill* and *taps*. These vectors may be specified by a vector containing a binary representation of the register and the taps, or by a list of locations for each cell set to one and each tap location. For example, an initial fill of [1 0 1 0 1] can also be specified by [1 3 5] and taps of [1 0 0 1 0] can be specified as [1 4]. If the binary representation is specified, its length must equal r . If a list is specified, its largest element must be less than r . The output is taken from a tap off cell one. Therefore, a tap off stage one is always included whether or not it is specified.

A maximal-length sequence can be generated using the appropriate taps. Consult a reference on coding for a table of taps that produce maximal-length sequences.

Example

Generate 15 bits of an R5(1,4) LRS with an initial fill of all ones.

```
y = lrs(5,15,[1 1 1 1 1],[4]);
y =
[1 0 0 1 1 0 1 0 0 1 0 0 0 0 1]
```



modindex

Purpose

Computes the modulation (deviation) index of an AM signal.

Synopsis

```
y = modindex(x);
```

Description

`modindex(x)` - Returns the modulation (deviation) index of *x*.

Note: If both the envelope AND modulation index are needed, the `envelope` command is more efficient.

Example

Return the modulation index of a `dsblc` modulated triangular wave used in the `dsblc` command example.

```
x = triwave(60,30);
y = dsblc(x,0.3,1024)
m = modindex(y);
m =
    3.0000
```

Algorithm

Finds the complex envelope and from its extrema computes:

$$m = \frac{\max - \min}{\max + \min}$$

See Also

`dsbsc`, `dsblc`, `envelope`

ook, ookmsg

Purpose

Generate an on-off keyed, bandpass signal.

Synopsis

```
y = ook(Rb,fc);
y = ook(Rb,fc,duration);
y = ook(Rb,fc,duration,fs);
y = ookmsg(Rb,fc,msg);
y = ookmsg(Rb,fc,fs,msg);
```

Description

`ook(Rb,fc)` - Generates one second of an on-off keyed signal with a bit rate of Rb bits-per-second centered at a carrier frequency, fc . Default sampling rate is 8192 Hz.

`ook(Rb,fc,duration)` - Generates *duration* seconds of the on-off keyed signal.

`ook(Rb,fc,duration,fs)` - Set the sampling frequency to fs .

`ookmsg(Rb,fc,msg)` or `ookmsg(Rb,fc,fs,msg)` - Generate a bandpass binary on-off keyed signal with a bit rate of Rb bits-per-second, at a carrier frequency, fc , sampled at sampling frequency, fs , and containing the message, msg . If fs is not given, fs defaults to 8192 Hz.

Example

Generate a 64 bit-per-second, on-off keyed signal at a carrier frequency of 2048 Hz lasting one-and-one-half seconds. Play the signal over the workstation's speaker.

```
y = ook(64,2048,1.5);
sound(y,8192);
```

Algorithm

The on-off keyed signal is generated by first creating a unipolar signal at the desired bit/sampling rates, and then applying a double-sideband, suppressed-carrier modulation to the resulting baseband signal at the desired carrier frequency.

Limitations

See limitations under `unipolar`.

See Also

`unipolar`, `dsbsc`, `bpsk`, `bpskmsg`, `bfsk`, `bfskmsg`

Reference

[1] Leon W. Couch, *Digital and Analog Communication Systems*, pp. 332-335, Macmillan, 1990.

proddmod

Purpose

Demodulate a bandpass digital signal using a product demodulator.

Synopsis

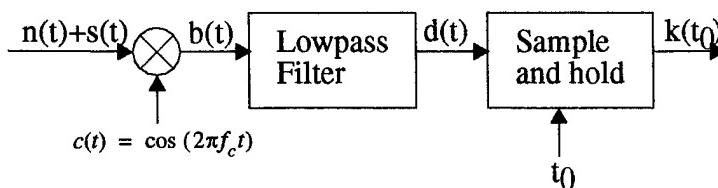
$y = \text{proddmod}(x, Rb, B, fc)$

$y = \text{proddmod}(x, Rb, B, fc, fs)$

Description

$\text{proddmod}(x, Rb, Fc)$ - Demodulates the bandpass digital signal x using a product demodulator [1]. A product demodulator is defined here as a sinusoidal mixer, followed by a low-pass filter, followed by a sample-and-hold block which is sub-sampled at the bit rate, $t_0 = Rb$. A cosine of fc Hz with zero-phase shift is used as the input to the sinusoidal mixer and a sixth order Chebychev Type 1 lowpass filter of bandwidth B is used. This detector is a coherent detector and thus requires the carrier used in modulating the bandpass digital signal to be a cosine wave with zero-phase shift. The return variable, y , is a vector containing the demodulated bit-stream. The default sampling frequency signal is $fs = 8192$ Hz.

$\text{proddmod}(x, Rb, fc, fs)$ - Sets the sampling frequency to f_s .



where $t_0 = 1/R_b$.

Example

Generate a BPSK signal containing the binary message [1 0 1 0 0 1] and demodulate using a correlation demodulator.

```

>> y = bpskmsg(64,1024,[1 0 1 0 0 1]);
>> proddmod(y,64,2*64,1024)
ans =
    1
    0
    1
    0
    0
    1
  
```

Algorithm

The input signal, $n(t)+s(t)$, is multiplied by $\cos(2\pi f_c t)$, reshaped to form a matrix where each column contains the samples for one bit period and then each column is summed to perform the integration. The integration output, $d(t)$, is then passed through

a threshold detector with the threshold set so that

$$k(t) = \begin{cases} 1 & k(t_0) \geq 0 \\ 0 & k(t_0) < 0 \end{cases}$$

Limitations

- The carrier frequency must be a multiple of the bit rate.
- If the number of samples-per-bit is not even, the last bit is zero padded. This can lead to an extra bit whose value is not based on samples from a full bit period and may be therefore in error.

See Also

corrdmod, bpsk, bpskmsg, bfsk, bfskmsg, ook, ookmsg

Reference

[1] Leon W. Couch, *Digital and Analog Communication Systems*, pp. 532-539, Macmillan, 1990.

sawwave

Purpose

Generate a baseband, sawtooth wave. This function can also be used to generate a ramp.

Synopsis

```
y = sawwave(fb)
y = sawwave(fb,'antipodal')
y = sawwave(fb,duration)
y = sawwave(fb,duration,'antipodal')
y = sawwave(fb,duration,fs)
y = sawwave(fb,duration,fs,'antipodal')
```

Description

`sawwave(fb)` - Generates one second of a baseband, sawtooth wave at cycle frequency, *fb*, and sampling frequency of 8192 Hz.

`sawwave(fb,duration)` - Generates *duration* seconds of a baseband, sawtooth wave at cycle frequency, *fb*, and sampling frequency of 8192 Hz.

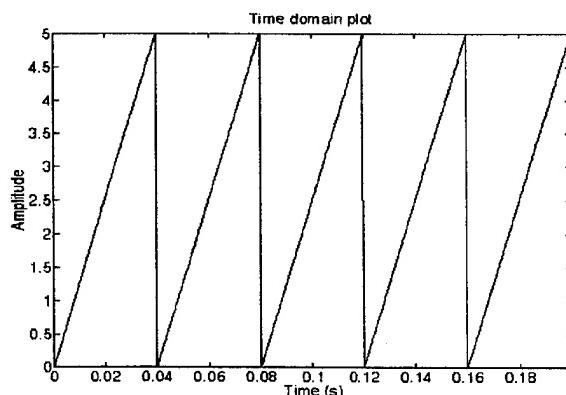
`sawwave(fb,duration,fs)` - Generates *duration* seconds of a baseband, sawtooth wave at cycle frequency, *fb*, and sampling frequency, *fs*.

The argument '*antipodal*' changes the amplitude range of the output waveform from [0,1] to [-1,1].

Examples

Generate 30 cycles of a sawtooth wave with a cycle frequency of 25 Hz sampled at 8000 Hz, varying between 0 and 5 volts.

```
x = 5 * sawwave(25,30/25,8000);
plottime(x,8000,0.2);
```



Generate a ramp from -1 to 3 with 100 samples:

```
y = 4 * sawwave(1,1,100) - 1;
```

Limitations

The waveform is created by first creating a single cycle of the waveform and then repeating this waveform the required number of times. Hence, when the cycle rate is not evenly divisible into the sampling frequency, truncation errors occur. The length of the output vector, *y*, can be computed by the formula:

$$\text{length} = \text{floor}(\text{duration} \cdot f_b) \cdot \text{floor}\left(\frac{f_s}{f_b}\right)$$

The exact cycle frequency can be computed from:

$$f_c = \frac{1}{\text{floor}\left(\frac{f_s}{f_b}\right)}$$

See Also

[sqwave](#), [trrowave](#), [coswave](#), [sinwave](#)

setsnr

Purpose

Mix two vectors (one representing a signal, one representing noise), such that the output signal's total bandwidth signal-to-noise ratio is set to the specified value.

Synopsis

```
[y,sigma] = setsnr(signal,SNR)
[y,sigma] = setsnr(signal,noise,SNR)
```

Description

`setsnr(signal,noise,SNR)` - Returns the signal total bandwidth signal-to-noise ratio equal to *SNR* decibels. The term "total bandwidth" implies the noise power in all frequencies from 0 to $fs/2$ is used in the computation of the SNR. Sigma is the noise variance described below.

`setsnr(signal,SNR)` - Gaussian white noise with zero-mean is added to the signal.

Example

Create a frequency-shift keyed signal and then add Gaussian white noise such that the total bandwidth SNR is 15 decibels.

```
x = bfsk(64,2048);
y = setsnr(x,randn(length(x),1),15);
```

Algorithm

The output signal is defined by

$$y[n] = s[n] + \sigma \cdot \omega[n]$$

for which σ is computed such that

$$SNR = 10 \log \left[\frac{Var(s[n])}{\sigma^2 Var(\omega[n])} \right],$$

where *SNR* is the desired signal-to-noise ratio. The amplitude of the noise is adjusted before the noise is added to produce the desired *SNR*. In the case of Gaussian white noise, relationship between σ and the noise power spectrum is given by:

$$\sigma^2 = \frac{N_0}{2}$$

See Also

`setsnrbw`

Reference

[1] Leon W. Couch, *Digital and Analog Communication Systems*, pp. 103-105, Macmillan, 1990.

setsnrbw

Purpose

Mix two vectors (one representing a signal, one representing noise), such that the output signal's noise-equivalent (in-band) signal-to-noise ratio is set to the specified value.

Synopsis

```
[y,sigma] = setsnrbw(signal,SNR,fc,bw)
[y,sigma] = setsnrbw(signal,noise,SNR,fc,bw,fs)
```

Description

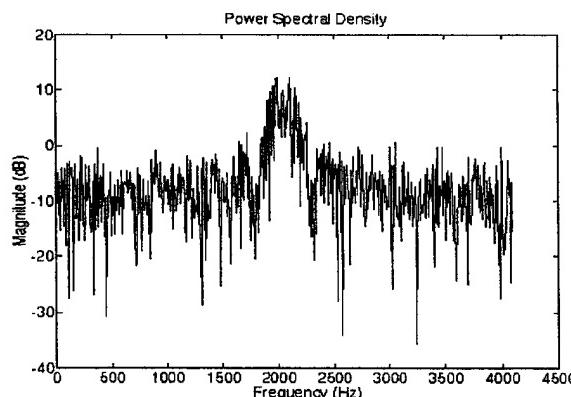
`setsnrbw(signal,noise,SNR,fc,bw,fs)` - Returns the signal with the noise-equivalent bandwidth (in-band) signal-to-noise ratio equal to *SNR* decibels. The term "noise-equivalent bandwidth" implies that only the power within the signal and noise specified bandwidth is used to compute the SNR. The bandwidth is centered at f_c and is equal to $2*bw$. This definition requires the value actually supplied for *bw* to be equal to one-half the desired bandwidth specification (see example). The sampling frequency, *fs*, is required to find the power spectral density of the noise. Sigma is the noise standard deviation described below.

`setsnrbw(signal,SNR,fc,bw)` - Gaussian white noise is added by the function and the sampling frequency defaults to 8192 Hz.

Example

Create a coherent, frequency-shift keyed signal and then add Gaussian white noise such that the noise-equivalent bandwidth SNR is 15 decibels. Use the distance from the center frequency to the first null in the bpsk signal spectrum as one-half the bandwidth.

```
x = bpsk(256,2048);
y = setsnrbw(x,15,2048,256);
lperigrm(y);
```



Algorithm

The output signal is defined by

$$y[n] = s[n] + \sigma \cdot \omega[n]$$

for which σ is computed such that

$$SNR = 10 \log \left[\frac{Var(s[n])}{\sigma^2 S_{BW}(\omega[n])} \right]$$

where SNR is the desired signal-to-noise ratio. The amplitude of the noise is adjusted before the noise is added to produce the desired SNR . In the case of Gaussian white noise, the relationship between σ and the power spectral density of the noise is:

$$\sigma^2 = \frac{N_0}{2}$$

See Also

[setsnr](#)

Reference

[1] Leon W. Couch, *Digital and Analog Communication Systems*, pp. 103-105, Macmillan, 1990.

sqwave

Purpose

Generate a baseband, unipolar [0,1] or antipodal [-1,1] square wave.

Synopsis

```
y = sqwave(fb)
y = sqwave(fb,'antipodal')
y = sqwave(fb,duration)
y = sqwave(fb,duration,'antipodal')
y = sqwave(fb,fs,nbrcycles)
y = sqwave(fb,fs,nbrcycles,'antipodal')
```

Description

sqwave(fb) - Generates one second of a baseband, square wave at cycle frequency, *fb*, and sampling frequency of *fs* = 8192 Hz.

sqwave(fb,duration) - Generates *duration* seconds of a baseband, square wave at cycle frequency, *fb*, and sampling frequency of 8192 Hz.

sqwave(fb,duration,fs) - Generates *duration* of a baseband unipolar [0,1] square wave at cycle frequency, *fb*, and sampling frequency, *fs*.

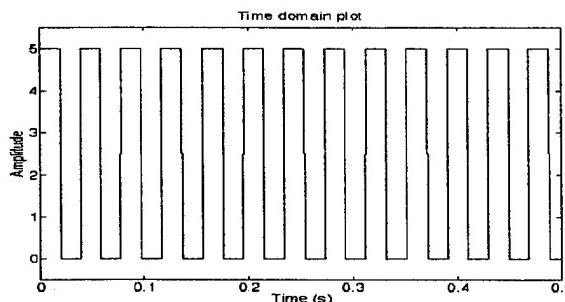
The argument '*antipodal*' changes the amplitude range of the output waveform from [0,1] to [-1,1].

Note: If the cycle frequency, *fb*, and the sampling frequency, *fs*, are both equal to one, the resulting vector will contain alternating values of [0,1] or [-1,1].

Examples

Generate 30 cycles of a square wave with a cycle frequency of 25 Hz sampled at 8000 Hz, varying between 0 and 5 volts.

```
x = 5 * sqwave(25,30/25,8000);
```



Generate 30 bits of a 50 bit-per-second signal sampled at 8000 Hz. Use this baseband signal as the modulating signal input to *dsbsc* to generate a phase-shift keyed signal.

```
x = sqwave(50/2,2*30/50,8000,'antipodal');
y = dsbsc(x,2000,8000);
```

Generate a vector with 100 elements alternating between 1 and 0.

```
y = sqwave(1,1,100)
y =
    1
    0
    1
    0
    1
    0
    1
```

(and so forth)

Limitations

The waveform is created by first computing a single cycle of the waveform and then by repeating this waveform the required number of times. Hence, when the cycle rate is not evenly divisible into the sampling frequency, truncation errors occur. The length of the output vector, *y*, can be computed by the formula:

$$\text{length} = \text{floor}(\text{duration} \cdot f_b) \cdot \text{floor}\left(\frac{f_s}{f_b}\right)$$

The exact cycle frequency can be computed from:

$$f_c = \frac{1}{\text{floor}\left(\frac{f_s}{f_b}\right)}$$

See Also

[antipodal](#), [unipolar](#), [trrowave](#), [sawwave](#)

str2masc, masc2str

Purpose

Convert a string into binary ASCII encoding (as sent by serial modems).

Synopsis

```
y = str2masc('string')
y = str2masc('string',databits,'parity',stopbits)
y = masc2str(x)
y = masc2str(x,databits,'parity',stopbits);
```

Description

`str2masc('string')` - Converts *string* to a vector of 1's and 0's representing raw, eight-bit ASCII coding.

`str2masc('string',databits,'parity',stopbits)` - Converts *string* to a vector of 1's and 0's representing modem encoded ASCII. The number of *databits* can be 7 or 8. *Parity* can be '*n*' for none, '*o*' for odd, or '*e*' for even. The number of *stopbits* can be 1 or 2. One start bit is always used. Valid combinations are:

7n1, 7e1, 7o1, 7n2, 7e2, 7o2, 8n1, 8n2

The output argument, *y*, is a $N \times 1$ vector where *N* is based upon the length of *string* and the combination of the parameters *bits*, *parity*, and *stopbits*.

`masc2str(x)` - Converts a vector of 1's and 0's representing raw eight-bit ASCII coding into a string.

`masc2str(x,databits,'parity',stopbits)` - Converts a vector of 1's and 0's representing the binary modem ASCII into a string. Valid argument combinations are the same as those listed above for `str2masc`. Parity and start/stop bit errors are detected and shown in the output string by the following special characters:

*, startbit error, ~, first stopbit error
, second stopbit error, +, parity error

Parity errors take precedence over start/stop bit errors in the output string.

Note: The bit stream produced meets the specifications of ANSI Standards X3.15 and X3.16 in that the start bit is a space, the stop bit(s) is a mark, and the data bits are ordered from least significant to most significant order [1].

Examples

Generate the binary representation of "BZ" sent at 7 bits-per-character with even parity and 2 stop bits. Convert back to a string.

```
y = str2masc('BZ',7,'e',2);
y =
[1 1 0 0 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 1 1]
x = masc2str(y,7,'e',2);
x =
BZ
```

Reference

[1] *The ARRL Handbook*, The American Radio Relay League, 1994, p. 19-18.

triwave

Purpose

Generate a baseband, triangular wave.

Synopsis

```
y = triwave(fb);
y = triwave(fb,'antipodal');
y = triwave(fb,duration);
y = triwave(fb,duration,'antipodal');
y = triwave(fb,duration,fs);
y = triwave(fb,duration,fs,'antipodal');
```

Description

`triwave(fb)` - Generates one second of a baseband, triangular wave at cycle frequency, fb , and sampling frequency of 8192 Hz.

`triwave(fb,duration)` - Generates $duration$ seconds of a baseband, triangular wave at cycle frequency, fb , and sampling frequency of 8192 Hz.

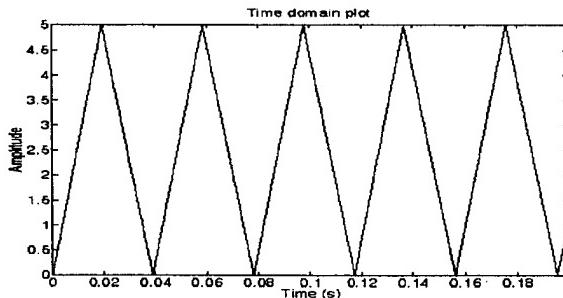
`triwave(fb,duration,fs)` - Generates $nbrcycles$ of a baseband, triangular wave at cycle frequency, fb , and sampling frequency, fs .

The argument '*antipodal*' changes the amplitude range of the output waveform from [0,1] to [-1,1].

Example

Generate 30 cycles of a triangular wave with a cycle frequency of 25 Hz sampled at 8000 Hz, varying between 0 and 5 volts.

```
x = 5 * triwave(25,30/25,8000)
```



Limitations

The waveform is created by first creating a single cycle of the waveform and then repeating this waveform the required number of times. Hence, when the cycle rate is not evenly divisible into the sampling frequency, truncation errors occur. The length of the output vector, y , can be computed by the formula:

$$\text{length} = \text{floor}(\text{duration} \cdot f_b) \cdot \text{floor}\left(\frac{f_s}{f_b}\right)$$

The exact cycle frequency can be computed from:

$$f_c = \frac{1}{\text{floor}\left(\frac{f_s}{f_b}\right)}$$

A “mathematically perfect” triangular wave is generated such that successive cycles fit perfectly. When the number of samples-per-cycle is odd, the maximum value of y will not be *exactly* 1, as occurs when the number of samples-per-cycle is even. In addition, the last cycle does not return *exactly* to zero in either case (the beginning of the *next* cycle would have started at zero). The “floor” term in the above equation is the number of samples-per-cycle.

See Also

`sqwave`, `sawwave`

unipolar

Purpose

Generate a baseband, unipolar [0,1] signal.

Synopsis

```
y = unipolar(fb,msg)
y = unipolar(fb,msg,fs)
```

Description

`unipolar(fb,msg)` - Generates a baseband unipolar signal at bit rate, fb . Sampling frequency defaults to 8192 Hz. If msg is a scalar, a *message* will be generated as a random binary sequence of length msg with $\text{Pr}(0)=\text{Pr}(1)=0.5$. If msg is a vector, it must be a vector of 0's and 1's. The length of the output vector, y , can be computed by the formula:

$$\text{length} = \text{nbrbaud} \cdot \text{floor}\left(\frac{f_s}{f_b}\right)$$

`unipolar(fb,msg,fs)` - Sets the sampling frequency to fs .

Note: If the bit rate, fb , and the sampling frequency, fs , are both equal to one, the resulting vector will contain alternating values of [0,1].

Example

Generate 30 bits of a 50 bit-per-second signal sampled at 8000 Hz. Use this baseband signal as the modulating signal input to `dsbsc` to generate an on-off keyed signal.

```
x = unipolar(50,30,8000);
y = dsbsc(x,2000,8000);
```

Limitations

- Due to the psuedo-random number generator, the probability of a one occurring 50% of the time will not be exact for short messages but will approach 50% as the message length increases.
- If the sampling frequency is not an exact multiple of the bit rate, unexpected results may occur. For example: `unipolar(9,18,20)` produces an output of 36 samples vice the expected 40 samples (i.e. $18 / 9 = 2$ seconds at 20 samples-per-second). This occurs since $\text{floor}(fs/fb) = 2$ samples-per-bit.

See Also

`antipodal`, `sqwave`

unipolar

Linear Function Reference

This section contains detailed descriptions of all linear functions in the Signal Processing and Communications Toolbox. It begins with a list of functions grouped by subject area and continues with the reference entries in alphabetical order. Information is also available through the online help facility.

Auto-Regressive/Moving-Average Modeling

<code>ar_corr</code>	Correlation AR method
<code>ar_covar</code>	Covariance AR method
<code>ar_mdcov</code>	Modified-covariance AR method
<code>ar_burg</code>	Burg AR method
<code>ar_levin</code>	Levinson recursion AR method
<code>ar_prony</code>	Prony MA method
<code>ar_shank</code>	Shank MA method
<code>ar_durbn</code>	Durbin MA method

Least Squares Modeling

<code>ls_whopf</code>	Wiener-Hopf method
<code>ls_svd</code>	Singular value decomposition method

Linear Systems

<code>minphase</code>	Minimum-phase polynomial
<code>maxphase</code>	Maximum-phase polynomial
<code>normaleq</code>	Normal equations solution

Correlation Matrix Estimation	
rxxcorr	No data pre- or post-windowing
rxxcovar	Data pre-windowing
rxxmdcov	Data pre- and post-windowing

Spectrum Estimation	
avgpergm	Average periodogram (Bartlett procedure)
blacktuk	Blackman Tukey procedure (correlogram)
daniell	Daniell periodogram (frequency smoothing)
freqeig	Frequency response transfer function for spectral estimates of the form $(e^{-jn\omega})^T \bar{A} (e^{jn\omega})$ where \bar{A} is a square matrix
minvarsp	Minimum variance (maximum likelihood) estimate
musicsp	MULTiple SIgnal Classification (MUSIC) estimate (can also be used to find the Pisarenko Harmonic Decomposition estimate)
musicspw	Eigenvector weighted MUSIC estimate
showeig	Eigenvalue magnitude plotting
specfact	Spectral factorization
welchsp	Welch procedure (overlapping, averaged periodograms)

Adaptive Filtering	
lmsale	Least mean square adaptive line enhancer filter

ar_burg

Purpose

Compute parameters for an AR model using the Burg method.

Synopsis

[a,b0,S,gamma,sigma] = ar_burg(x,P)

Description

ar_burg(x,P) - Computes the coefficients of an order P Auto-Regressive Model (AR) of the data given in vector x via the Burg method as put forth in [1]. The output is specified by:

a = filter coefficients
 b_0 = gain
 S = prediction error variance
 gamma = reflection coefficients
 sigma = sum-of-squared errors

The gain term, b_0 , is not computed in the Burg algorithm itself. It is computed using "brute force" by generating the impulse response of the model and then applying:

$$\text{gain} = \sqrt{\frac{P_x}{P_{\text{model}}}}$$

Example

Estimate the parameters for a second order AR model for the data sequence {1, -2, 3, -4, 5}. (Example 9.6 in [1].)

```

>> [a,b0,S] = ar_burg([1 -2 3 -4 5],2)
a =
  1.0000  1.8639  0.9571
b0 =
  1.3934
S =
  0.0859
  
```

Algorithm

The recursion specified by the following equations [1]:

$$a_p = \begin{bmatrix} a_{p-1} \\ \emptyset \end{bmatrix} - \gamma_p^* \begin{bmatrix} \emptyset \\ \tilde{a}_{p-1}^* \end{bmatrix} \quad \sigma_{\varepsilon_p}^2 = (1 - |\gamma_p|^2) \sigma_{\varepsilon_{p-1}}^2$$

See Also

ar_corr, ar_covar, ar_mdcov, ar_prony, ar_durbn, ar_shank, ar_levin

Reference

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 545-548, Prentice-Hall, 1992.

ar_corr

Purpose

Compute parameters for an AR model using the autocorrelation method.

Synopsis

[*a*,*b0*,*S*,*s*] = ar_corr(*x*,*P*)

Description

ar_corr(*x*,*P*) - Computes the coefficients of an order *P* Auto-Regressive Model (AR) of the data given in vector *x* via the autocorrelation method as put forth in [1]. The output is specified by:

a = filter coefficients

b0 = gain

S = minimum sum of squares

s = estimate for prediction error variance

Example

Estimate the parameters for a second order AR model for the data sequence {1, -2, 3, -4, 5}. (Example 9.6 in [1].)

```
>> [a,b0,S,s] = ar_corr([1 -2 3 -4 5],2)
a =
    1.0000   0.8140   0.1193
b0 =
    5.0537
S =
    25.5404
s =
    3.6486
```

Algorithm

$$b_0 = \sqrt{S} \quad X^H X a = \begin{bmatrix} S \\ \emptyset \end{bmatrix}$$

See Also

ar_burg, ar_covar, ar_mdcov, ar_prony, ar_durbn, ar_shank, ar_levin

Reference

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 535-541, Prentice-Hall, 1992.

ar_covar

Purpose

Compute parameters for an AR model using the covariance method.

Synopsis

[a,b0,S,s] = ar_covar(x,P)

Description

ar_covar(x,P) - Computes the coefficients of an order P Auto-Regressive Model (AR) of the data given in vector x via the covariance method as put forth in [1]. The output is specified by:

a = filter coefficients
 b_0 = gain
 S = minimum sum of squares
 s = estimate for prediction error variance

The gain term, b_0 , is not computed in the covariance algorithm itself. It is computed using brute force by generating the impulse response of the model and then applying:

$$\text{gain} = \sqrt{\frac{P_x}{P_{\text{model}}}}$$

Example

Estimate the parameters for a second order AR model for the data sequence {1, -2, 3, -4, 5}. (Example 9.6 in [1].)

```
>> [a,b0,S,s] = ar_covar([1 -2 3 -4 5],2)
a =
    1.0000  2.0000  1.0000
b0 =
    1.0000
S =
    9.3792e-13
s =
    1.8758e-13
```

Algorithm

$$(X^T X) a = \begin{bmatrix} S \\ \emptyset \end{bmatrix}$$

See Also

ar_burg, ar_corr, ar_mdcov, ar_prony, ar_durbn, ar_shank, ar_levin

Reference

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 535-541, Prentice-Hall, 1992.

ar_durbn

Purpose

Compute parameters for an ARMA model using the Durbin method.

Synopsis

[*a,b*] = ar_durbn(*x,P,Q,L*)

[*a,b*] = ar_durbn(*x,P,Q,L,'armethod'*)

Description

ar_durbn(*x,P,Q,L*) - Computes the coefficients of an order *P* Auto-Regressive, order *Q* Moving Average Model of the data given in vector *x*. The *L* parameter can be used to set the order of the intermediate AR model used in the algorithm. If not provided, *L* defaults to $5 * Q$ or the length of *x minus 1*, whichever is smaller. The intermediate AR model is first found using the covariance method. This model is then used to generate new data from which the MA parameters are found using the Durbin method. The AR parameters are found using the covariance method. The output arguments are:

a = AR coefficients

b = MA coefficients

ar_durbn(*x,P,Q,'armethod'*) and ar_durbn(*x,P,Q,L,'armethod'*) - Uses the AR method specified by '*armethod*' to compute the AR parameters and the intermediate AR model during computation of the Durbin MA parameters. Valid AR methods are '*ar_corr*', '*ar_covar*', '*ar_mdcov*' and '*ar_burg*'.

Limitations

The Durbin method is not well suited for small data sets.

See Also

ar_burg, ar_corr, ar_covar, ar_mdcov, ar_prony, ar_shank, ar_levin

Reference

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 558-560, Prentice-Hall, 1992.

ar_levin

Purpose

Compute parameters for an AR model using the Levinson recursion.

Synopsis

```
[a,s]=ar_levin(r,P)
```

Description

`ar_levin(r,P)` - Computes the coefficients of an order P Auto-Regressive Model from values of the autocorrelation vector $r = [R(0) R(1) R(2) \dots R(P)]$. The autocorrelation must be symmetric ($R(1) = R^*(-1)$, $R(2) = R^*(-2)$, etc.,). The output arguments are:

a = AR parameters
 s = linear prediction error variance

Example

Given $r = [3 2 1 0 0 0 \dots]$, find the second order AR model. (Example 8.1 in [1])

```
>> [a,s] = ar_levin([3 2 1 0],2)
a =
    1.0000   -0.8000   0.2000
s =
    1.6000
```

Algorithm

Initialization:

$$r_0 = R_x[1] \quad a_0 = 1 \quad \sigma^2 = R_x[0]$$

Recursion:

$$\begin{aligned} \Delta_p &= \mathbf{r}_{p-1}^{*T} \tilde{\mathbf{a}}_{p-1} \\ \gamma_p &= \frac{\Delta_p}{\sigma_{p-1}^2} \\ \mathbf{a}_p &= \begin{bmatrix} \mathbf{a}_{p-1} \\ \emptyset \end{bmatrix} - \gamma_p \begin{bmatrix} \emptyset \\ \tilde{\mathbf{a}}_{p-1}^* \end{bmatrix} \\ \sigma_p &= \sigma_{p-1}^2 - \gamma_p \Delta_p^* \end{aligned}$$

See Also

`ar_burg`, `ar_corr`, `ar_covar`, `ar_mdcov`, `ar_prony`, `ar_durbn`, `ar_shank`

Reference

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 422-430, Prentice-Hall, 1992.

ar_mdcov

Purpose

Compute parameters for an AR model using the modified covariance method.

Synopsis

[a,b0,S,s] = ar_mdcov(x,P)

Description

ar_mdcov(x,P) - Computes the coefficients of an order P Auto-Regressive Model (AR) of the data given in vector x via the modified covariance method as put forth in [1].

The output is specified by:

a = filter coefficients

b0 = gain

S = minimum sum of squares

s = estimate for prediction error variance

The gain term, b_0 , is not computed in the modified covariance algorithm itself. It is computed using brute force by generating the impulse response of the model and then applying:

$$gain = \sqrt{\frac{P_x}{P_{model}}}$$

Example

Estimate the parameters for a second order AR model for the data sequence {1, -2, 3, -4, 5}. (Example 9.6 in [1].)

```
>> [a,b0,S,s] = ar_mdcov([1 -2 3 -4 5],2)
a =
    1   2   1
b0 =
    1
S =
    0
s =
    0
```

Algorithm

$$(X^T X + \tilde{X}^T \tilde{X}^*)_a = \begin{bmatrix} S \\ \emptyset \end{bmatrix}$$

See Also

[ar_burg](#), [ar_corr](#), [ar_covar](#), [ar_prony](#), [ar_durbn](#), [ar_shank](#), [ar_levin](#)

Reference

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 542-544, Prentice-Hall, 1992.

ar_prony

Purpose

Compute parameters for an ARMA model using the Prony method.

Synopsis

[*a,b*] = ar_prony(*x,P,Q*)

Description

ar_prony(*x,P,Q*) - Computes the coefficients of an order *P* Auto-Regressive, order *Q* Moving Average Model of the data given in vector *x*. The AR parameters are found using the covariance method. MA parameters are found using the Prony method. The output arguments are:

a = AR coefficients

b = MA coefficients

ar_prony(*x,P,Q,'armethod'*) - Uses the AR method specified by '*armethod*' to compute the AR parameters. Valid AR methods are '*ar_corr*', '*ar_covar*', '*ar_mdcov*' and '*ar_burg*'.

See Also

ar_burg, *ar_corr*, *ar_covar*, *ar_mdcov*, *ar_durbin*, *ar_shank*, *ar_levin*

Reference

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 550-555, Prentice-Hall, 1992.

ar_shank

Purpose

Compute parameters for an ARMA model using the Shank method.

Synopsis

[*a,b*] = ar_shank(*x,P,Q*)

Description

ar_shank(*x,P,Q*) - Computes the coefficients of an order *P* Auto-Regressive, order *Q* Moving Average Model of the data given in vector *x*. The AR parameters are found using the covariance method. MA parameters are found using the Shank method. The output arguments are:

a = AR coefficients

b = MA coefficients

ar_shank(*x,P,Q,'armethod'*) - Uses the AR method specified by '*armethod*' to compute the AR parameters. Valid AR methods are '*ar_corr*', '*ar_covar*', '*ar_mdcov*' and '*ar_burg*'.

See Also

ar_burg, *ar_corr*, *ar_covar*, *ar_mdcov*, *ar_prony*, *ar_durbn*, *ar_levin*

Reference

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 555-558, Prentice-Hall, 1992.

avgpergm

Purpose

Compute the averaged periodogram spectral estimate (Bartlett procedure).

Synopsis

```
[Pxx,stddev] = avgpergm(x)
[Pxx,stddev] = avgpergm(x,N)
[Pxx,stddev] = avgpergm(x,N,window)
[Pxx,stddev] = avgpergm(x,N,'frames')
[Pxx,stddev] = avgpergm(x,N,window,'frames')
```

Description

$[P_{xx},\text{stddev}] = \text{avgpergm}(x)$ - Computes the averaged, non-overlapping periodogram (Bartlett procedure) of signal x , the magnitude is returned in P_{xx} and the standard deviation in stddev . The units of the returned vector are the same as the FFT.

$[P_{xx},\text{stddev}] = \text{avgpergm}(x,N)$ - Sets the length of the FFT used to N .

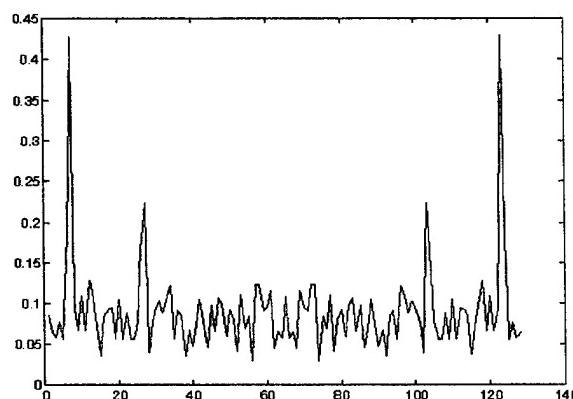
$[P_{xx},\text{stddev}] = \text{avgpergm}(x,N,\text{window})$ - Uses the N -point data smoothing window.

$[P_{xx},\text{stddev}] = \text{avgpergm}(x,N,\text{'frames'})$ or $[P_{xx},\text{stddev}] = \text{avgpergm}(x,N,\text{window},\text{'frames'})$ returns a N by L matrix where each column in X is a single frame in the periodogram and L is the number of window dependent upon the length of x and the size of N . In this use of the command, the average periodogram is not returned but can be found from P_{xx} by averaging across the columns of P_{xx} .

Example

Estimate the spectrum of two sinusoidal signals in Gaussian white noise.

```
>> t = 0:0.001:0.5;
>> s = cos(2*pi*45*t) + 3/4 * sin(2*pi*200*t);
>> s = s + randn(1,length(s));
>> [pxx] = avgpergm(s,128);
>> plot(pxx)
```



Algorithm

The averaged periodogram spectral estimate is given by:

$$\hat{S}(e^{j\omega}) = \frac{1}{K} \sum_{k=1}^K P_x^{(k)}(e^{j\omega})$$

where $P_x^{(k)}$ is the periodogram of the k^{th} data segment of length N

$$P_x^{(k)}(e^{j\omega}) = \frac{1}{N} |X^{(k)}(e^{j\omega})|^2$$

and

$$X^{(k)}(e^{j\omega}) = \sum_{n=0}^{N-1} x(n) w_k(n) e^{-j\omega n}$$

where $w_k(n)$ is the data smoothing window associated with the k^{th} data segment. No overlap between frames is used.

See Also

[blacktuk](#), [welchsp](#)

Reference

- [1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 586-596, Prentice-Hall, 1992.

blacktuk

Purpose

Compute the Blackman-Tukey spectral estimate (correlogram).

Synopsis

[Pxx] = blacktuk(x)

[Pxx, stddev] = blacktuk(x, N)

[Pxx, stddev] = blacktuk(x, N, 'frames')

Description

[Pxx] = blacktuk(x) - Computes the Blackman-Tukey spectral estimate of *x*. The magnitude is returned in *Pxx*. The size of the FFT applied to the auto-correlation function is given by the equation

$$N = 2^{\lceil \log_2(2\text{length}(x) - 1) \rceil}$$

that is, the next power-of-two greater-than or equal-to the length of the auto-correlation). A Bartlett (triangular) window is applied to the autocorrelation function.

[Pxx, stddev] = blacktuk(x, N) - Sets the length of the FFT used to *N*. The data is framed so that *N*-1 data points are used in computing the auto-correlation. The auto-correlation of the individual data frames is averaged to produce the final estimate. Only full frames of data are used (any excess data is thrown away). A Bartlett window is applied to the autocorrelation function. The standard deviation of the frames used in the estimate is returned in *Pxx*. No overlap between frames is used.

[Pxx] = blacktuk(x, N, 'frames') - Returns a *N* by *K* matrix where each column in *Pxx* is a single frame in the correlogram. The averaged estimate is not returned.

Example

Estimate the spectrum of two sinusoidal signals in Gaussian white noise using the Blackman-Tukey procedure.

```
>> t = 0:0.001:0.5;
>> s = cos(2*pi*45*t) + 3/4 * sin(2*pi*200*t);
>> s = s + randn(1,length(s));
>> [pxx] = blacktuk(s, 128);
>> plot(pxx)
```

Algorithm

The correlogram is given by

$$\hat{S}_x(e^{j\omega}) = \sum_{l=-L}^L R_x[l] e^{-j\omega l}$$

for $L < N_s$ where $2L$ is the length of the window and where the biased estimate for of the auto-correlation is specified by

$$R_x[l] = \frac{1}{N} \sum_{n=0}^{N-1-l} x[n+l] x^*[n]$$

for $0 \leq l < N$.

See Also

`avgpergm`, `daniel`, `welchsp`

daniell

Purpose

Compute Daniell periodogram.

Synopsis

$X = \text{daniell}(x,k)$

$X = \text{daniell}(x,k,nfft)$

Description

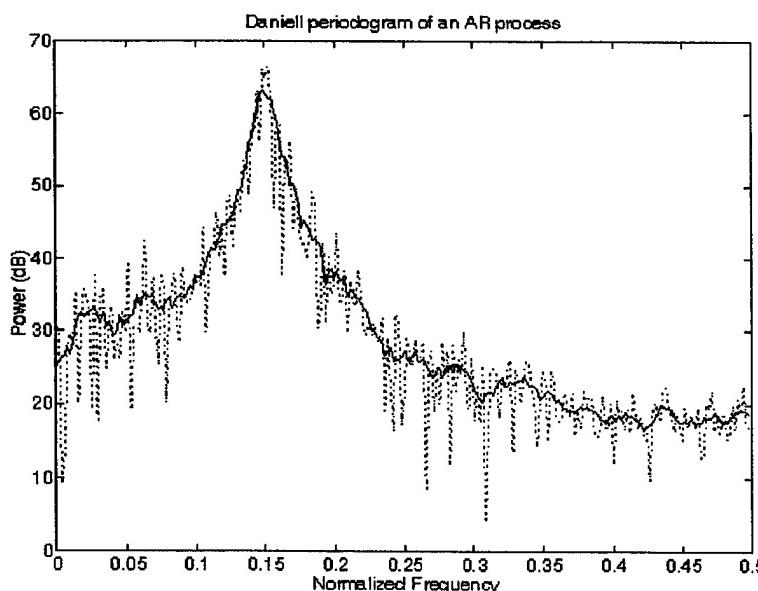
$X = \text{daniell}(x,k)$ - Computes the Daniell periodogram of x averaging k adjacent frequency bins of the FFT. If k is even, it is rounded up one to make it odd.

$X = \text{daniell}(x,k,nfft)$ - Sets the FFT length to $nfft$ points. The default FFT length is the number of samples in x .

Example

Compute the Daniell periodogram averaging seven adjacent frequency bins of the signal contained in an AR process.

```
a = [1.0000 -1.9120 2.4218 -1.4066 0.5656];
y = filter(1,a,randn(512,1));
Xfft = 20*log10(abs(fft(y)));
Xdan = 20*log10(abs(daniell(y,7)));
fscale = (0:255)/512;
plot(fscale,Xfft(1:256),':',fscale,Xdan(1:256),'-')
```



Algorithm

The Daniell periodogram is obtained by averaging P adjacent frequency bins about a given frequency bin, f_i , of the FFT [1]. P is computed as $P = (k - 1)/2$. The Daniell periodogram is given by the equation

$$\hat{P}_D = \frac{1}{2P+1} \sum_{n=i-P}^{i+P} \tilde{P}_{xx}(f_n)$$

where \tilde{P}_{xx} is the periodogram given by

$$\tilde{P}_{xx}(f) = \frac{1}{N} \left| \sum_{n=0}^{N-1} x(n) e^{-j2\pi fn} \right|^2$$

The function `avsmooth` is used as the averaging filter.

See Also

`avsmooth`, `spect2d`

Reference

[1] S. Lawrence Marple, *Digital Spectral Analysis with Applications*, p. 153, Prentice-Hall, 1987.

freqeig

Purpose

Compute frequency response on spectral estimators in the form $(e^{-jn\omega})^T \bar{A} (e^{jn\omega})$.

Synopsis

[h,f] = freqeig(a,N)

[h,f] = freqeig(a,N,fs)

Description

[h,f] = freqeig(a,N,fs) - Evaluates the frequency response of a spectral estimate in the form

$$Q = w^H \bar{A} w,$$

where

$$w = \begin{bmatrix} 1 \\ e^{j\omega} \\ e^{j2\omega} \\ \dots \\ e^{j(N-1)\omega} \end{bmatrix},$$

and A is a Hermitian symmetric matrix. The algorithm exploits the conjugate symmetric property of the resulting polynomial to compute the frequency response using a single FFT of size N. If the sampling frequency, fs, is not given, the frequency vector f is returned such that 1 equals the Nyquist frequency.

See Also

minvarsp, musicsp, musicspw

lmsale

Purpose

Least mean square adaptive line enhancer filter.

Synopsis

```
[w,y,e] = lmsale(x,m,step,delay)
[w,y,e] = lmsale(x,win,step,delay)
```

Description

[w,y,e] = lmsale(x,m,step,delay) implements an adaptive line enhancer using the least-mean squares approach where x is the input signal, m is the desired number of filter weights (order), step is the percentage of the maximum step size to use in computing the next set of filter weights and delay is the number of samples to delay x in computing the new weights. The final filter weights are returned in w, the estimated signal is returned in y, and the error signal is returned in e.

The maximum step size is computed from

$$maxstep = \frac{2/m}{std(x)^2}$$

[w,y,e] = lmsale(x,win,step,delay) uses the vector win as the initial guess at the weights. The number of weights, m, is equal to the length of win.

Examples

Create a 60 Hertz tone in noise and then use an adaptive line enhancer filter to recover the tone.

```
t = (0:1000) / 1000;
s = sin(2 * pi * 60 .* t);
sn = s + randn(1,1001);
[w,y,e] = lmsale(sn,8,5,4);
subplot(2,1,1),plot(sn),subplot(2,1,2),plot(y)
```

Limitations

This function is implemented as both an m-file and a mex-file. The mex-file version is several times faster than the m-file implementation. To compile the mex file under UNIX, use the command "cmex -D__UNIX__ lmsale.c" while in the spcline subdirectory.

See Also

aledsgn

Reference

[1] Simon Haykin, *Adaptive Filter Theory*, pp 44-48, 299-304, Prentice Hall, 1991.

ls_svd

Purpose

Compute the filter coefficients for a least-squares optimal filter using singular value decomposition.

Synopsis

[h,S,Px] = ar_svd(d,x,P)

Description

ar_svd(d,x,P) - Given the observed data sequence x , compute the filter coefficients for a least-squares optimal filter of order P , that produce the desired data sequence d from the observed data sequence. Output arguments are:

h = filter coefficients
 S = sum of squared errors
 Px = the projection matrix

Example

Given the desired data sequence [1 -2 3 -4 5], compute the filter coefficients of the second order least-squares filter that will produce the desired data sequence [1 -1 1 -1 1]. (Example 9.4 from [1])

```
>> d = [1 -1 1 -1 1];
>> x = [1 -2 3 -4 5];
>> [h,S] = ls_svd(d,x,2)
h =
    1.0000
    1.0000
S =
    5.3291e-15
>> y = filter(1,h,[1 0 0 0 0])
y =
    1.0000   -1.0000    1.0000   -1.0000    1.0000
```

See Also

ls_whopf

Reference

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 518-528, Prentice-Hall, 1992.

ls_whopf

Purpose

Compute the filter coefficients for a least-squares optimal filter using the Wiener-Hopf equation.

Synopsis

[*h,S,Px*] = ar_whopf(*d,x,P*)

Description

ar_whopf(*d,x,P*) - Given the observed data sequence *x*, compute the filter coefficients for a least-squares optimal filter of order *P*, that will produce the desired data sequence *d* from the observed data sequence. Output arguments are:

h = filter coefficients
S = sum of squared errors
Px = projection matrix

Example

Given the desired data sequence [1 -2 3 -4 5], compute the filter coefficients of the second order least-squares filter that will produce the desired data sequence [1 -1 1 -1 1]. (Example 9.3 from [1])

```
>> d = [1 -1 1 -1 1];
>> x = [1 -2 3 -4 5];
>> [h,S] = ls_whopf(d,x,2)
h =
    1.0000
    1.0000
S =
    -1.7764e-14
>> y = filter(1,h,[1 0 0 0 0])
y =
    1.0000   -1.0000   1.0000   -1.0000   1.0000
```

Algorithm

Wiener-Hopf equation solved for *h*.

$$= (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{d}$$

Sum of squared errors:

$$S = \mathbf{d}^T \mathbf{d} - \mathbf{d}^T \mathbf{X} \mathbf{h}$$

See Also

ls_svd

Reference

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 518-528, Prentice-Hall, 1992.

minphase, maxphase

Purpose

Return a polynomial that is in minimum or maximum phase form.

Synopsis

$y = \text{minphase}(P)$

$y = \text{maxphase}(P)$

Description

$\text{minphase}(P)$ - Returns a minimum phase form polynomial for P if P is not minimum phase already. If P already is minimum phase, P is simply returned. A minimum phase polynomial is one whose roots are all inside the unit circle.

$\text{maxphase}(P)$ - Returns a maximum phase form polynomial for P if P is not maximum phase already. If P already is maximum phase, P is simply returned. A maximum phase polynomial is one whose roots are all outside the unit circle.

$\text{minphase}(P,\text{tol})$ and $\text{maxphase}(P,\text{tol})$ - Set the tolerance used in pairing complex roots using the `cplxpair` command. Default tolerance is 0.0001.

Example

Return the minimum phase form of $2z^2 + z + 4$. The roots of this polynomial are:

$$r_{1,2} = 1.414e^{\pm j\frac{\pi}{4}}$$

The roots of the resulting minimum phase polynomial are:

$$r_{1,2} = 0.7071e^{\pm j\frac{\pi}{4}}$$

It can be verified using the `freqz` command that the magnitude of the impulse response for both these polynomials are the same and that the second polynomial has minimum phase lag.

```
>> b = minphase([2 1 4])
b =
    4.0000   1.0000   2.0000
[h,w] = freqz(b,1,50);
mag = abs(h); phase = angle(h);
semilogw(w,mag); semilogw(w,phase);
```

Polynomial b is $4z^2 + z + 2$.

Algorithm

The algorithm for `minphase`, in essence, multiplies the polynomial by the following factor for each root that has a magnitude greater than 1.

$$p = p \frac{zz_0^* - 1}{z_0 - z}$$

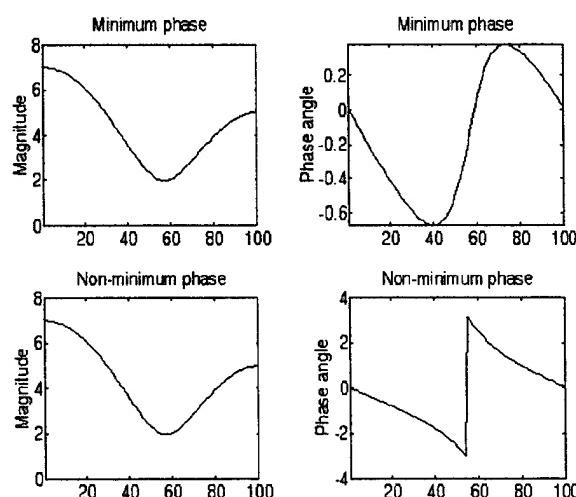
The function `minphase` actually computes the minimum phase polynomial using the

minphase

following steps.

1. The first non-zero coefficient is found and factored from of the polynomial. The coefficient is saved in order to compute the gain of the minimum phase polynomial.
2. The roots of the polynomial are computed and paired.
3. The roots of the polynomial located outside the unit circle are found.
4. The magnitude of these roots are computed and multiplied by the gain term factored in step 1 to form a new gain term.
5. The inverse conjugate of the roots lying outside the unit circle are found and recombined with the roots lying inside the circle to form a minimum phase polynomial.
6. The resulting polynomial is multiplied by the gain term to ensure the magnitude of the minimum phase polynomial equals the magnitude of the mixed phase polynomial.

The function **maxphase** works similar to **minphase**, only in a reciprocal fashion.



References

- [1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 250-253, Prentice-Hall, 1992.
- [2] Alan V. Oppenheim & Ronald W. Schafer, *Digital Signal Processing*, pp 345-353, Prentice-Hall, 1975.

minvars

Purpose

Minimum variance (“maximum likelihood”) spectral estimation.

Synopsis

[h,a]=minvars(Rx,n)

Description

[h,a]=minvars(rx,n) computes the spectral estimate of the signal whose correlation matrix is given in the variable R_x and returns the n-point magnitude of the spectral estimate in h. The correlation matrix, R_x , must be a square matrix. The return argument a is the transfer function denominator polynomial of degree 2^N-1 .

[h,a]=minvars(x,P,n) computes the spectral estimate of the signal x. The function computes a PxP correlation matrix using the 'rxxcovar' method.

[h,a]=minvars(x,p,n,'rxxmethod') computes the PxP correlation matrix using the method prescribed by 'rxxmethod'. Valid methods are:

- 'rxxcorr' - AR auto-correlation estimate method.
- 'rxxcovar' - AR covariance estimate method.
- 'rxxmdcov' - AR modified-covariance estimate method.

The 'rxxcovar' option is the default if no method is given.

Examples

Compute the spectral estimate of the signal datac using a 5 x 5 correlation matrix based on the modified covariance method.

```
[h,a] = minvars(datac,5,256);
fs = 1; f = (0:255)/fs/2;
plot(f,h)
```

Algorithm

The minimum variance (maximum likelihood) spectral estimate is computed from

$$\hat{S}_{ML}(e^{j\omega}) = \frac{1}{\mathbf{w}^* \mathbf{R}_x^{-1} \mathbf{w}}$$

where

$$\mathbf{w} = \begin{bmatrix} 1 \\ e^{j\omega} \\ \dots \\ e^{j(N-1)\omega} \end{bmatrix}$$

minvars

See Also

rxxcorr, *rxxcovar*, *rxxmdcov*

References

- [1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 607-614, Prentice-Hall, 1992.

musicsp, musicspw

Purpose

MUltiple Signal Classification (MUSIC) spectral estimation.

Synopsis

```
[h,a]=musicsp(Rx,n)
[h,a]=musicsp(Rx,q,n)
[h,a]=musicsp(x,p,q,n)
[h,a]=musicsp(x,p,q,n,'rxxmethod')
```

Description

[h,a]=musicsp(rx,n) computes the MUSIC psuedo-spectral estimate of the signal whose correlation matrix is given in the variable Rx, returning the n-point magnitude of the psuedo-spectral estimate in h. The correlation matrix, Rx, must be a square matrix. This form of the function assumes the number of complex sinusoids contained in the signal is one less than the rank of correlation matrix Rx. Return argument a is the "root" MUSIC polynomial.

Note: A real sinusoid contains two complex sinusoids.

[h,a]=musicsp(Rx,q,n) forces the algorithm to assume there are q complex sinusoidal signals. The variable q must be at least one less than the rank correlation matrix Rx.

[h,a]=musicsp(x,p,q,n) computes the MUSIC psuedo-spectral estimate of the signal x. The function computes a p x p correlation matrix using the 'rxxcovar' function and assumes p-1 complex sinusoids.

[h,a]=musicsp(x,p,q,n,'rxxmethod') computes the p x p correlation matrix using the method prescribed by 'rxxmethod'. Valid options are:

- 'rxxcorr' - AR auto-correlation estimate method.
- 'rxxcovar' - AR covariance estimate method.
- 'rxxmdcov' - AR modified-covariance estimate method.

The 'rxxcovar' option is the default if no method is given.

Note: Setting the rank of the projection matrix, q, to one less than number of complex sinusoids, p, computes a spectral estimate based on the Pisarenko harmonic decomposition.

The musicspw function takes the same forms as musicsp and computes the weighted MUSIC pseudo-spectrum (eigenvectors used are weighted by 1/eigenvalue).

Examples

Compute the spectral estimate of the signal datac using a 5 x 5 correlation matrix based on the modified covariance method.

```
[h,a] = musicsp(datac,5,256);
fs = 1; f = (0:255)/fs/2;
plot(f,h)
```

Algorithm

The MUSIC psuedo-spectral estimate is computed from

$$P_{MU}(e^{j\omega}) = \frac{1}{\mathbf{w}^* \mathbf{P}_{noise} \mathbf{w}} = \frac{1}{\mathbf{w}^* \mathbf{E}_{noise} \mathbf{E}_{noise}^* \mathbf{w}}$$

where

$$\mathbf{w} = \begin{bmatrix} 1 \\ e^{j\omega} \\ \dots \\ e^{j(N-1)\omega} \end{bmatrix}$$

and \mathbf{P}_{noise} is the projection matrix of the signal onto the noise subspace. For more information, see [1].

See Also

`rxxcorr`, `rxxcovar`, `rxxmdcov`

References

- [1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 626-630, Prentice-Hall, 1992.

normaleq

Purpose

Solves a system of Normal equations used in linear predictive filtering.

Synopsis

[e,a] = normaleq(R)

Description

normaleq(R) - Returns the prediction error variance, e , and the linear predictive filter coefficients, $\begin{bmatrix} 1 & a_1 & a_2 & \dots & a_p \end{bmatrix}$ of the correlation matrix, R .

Example

Use normaleq to solve the system of normal equations when $R_x[0]=1$, $R_x[1]=R_x[-1]=0.5$, $R_x[2]=R_x[-2]=0.25$ (Example 7.1 of [1]).

```
>> R = toeplitz([1 0.5 0.25])
R =
    1.0000    0.5000    0.2500
    0.5000    1.0000    0.5000
    0.2500    0.5000    1.0000
>> [e,a] = normaleq(R)
e =
    0.7500
a =
    -0.5000
    0
```

Algorithm

This function is a direct implementation of equation (7.23) given in [1]. Refer to [1] for a detailed discussion. In references on linear predictive filtering, these equations are often referred to as the *augmented normal equations*.

Normal equations for $p = 2$:

$$\begin{bmatrix} R_x[0] & R_x[1] & R_x[2] \\ R_x^*[1] & R_x[0] & R_x[1] \\ R_x^*[2] & R_x^*[1] & R_x[0] \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sigma_e^2 \\ 0 \\ 0 \end{bmatrix}$$

where σ_e^2 is the prediction error variance and $R_x[n]$ is the auto-correlation of x at n .

Reference

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, eq 7.23, p 345, Prentice-Hall, 1992.

rxxcorr, rxxcovar, rxxmdcov

Purpose

Estimate the autocorrelation matrix.

Synopsis

```
rxx = rxxcorr(x,p)  
rxx = rxxcovar(x,p)  
rxx = rxxmdcov(x,p)
```

Description

rxxcorr(x,p) estimates a $P \times P$ auto-correlation matrix for the data in *x* using the method of estimating the auto-correlation matrix used in the auto-correlation method of AR modeling. The resulting auto-correlation matrix will be Hermitian.

rxxcovar(x,p) estimates a $P \times P$ auto-correlation matrix for the data in *x* using the method of estimating the auto-correlation matrix used in the covariance method of AR modeling. The resulting auto-correlation matrix will be Hermitian but not Toeplitz.

rxxmdcov(x,p) estimates a $P \times P$ auto-correlation matrix for the data in *x* using the method of estimating the auto-correlation matrix used in the modified covariance method of AR modeling. The resulting auto-correlation matrix will not be Toeplitz.

Examples

Estimate the autocorrelation matrix of the data [1 -2 3 -4 5].

```
x = [1 -2 3 -4 5];  
rxx = rxxcorr(x,3)  
rxx =  
    55  -40   26  
   -40   55  -40  
    26  -40   55  
  
rxx = rxxcovar(x,3)  
rxx =  
    50  -38   26  
   -38   29  -20  
    26  -20   14  
  
rxx = rxxmdcov(x,3)  
rxx =  
    64  -58   52  
   -58   58  -58  
    52  -58   64
```

Algorithm

Examples of the data matrix used to form the correlation matrix estimates are shown below for N=5 datapoints.

Auto-correlation method.

$$X = \begin{bmatrix} x[0] & 0 & 0 \\ x[1] & x[0] & 0 \\ x[2] & x[1] & x[0] \\ x[3] & x[2] & x[1] \\ x[4] & x[3] & x[2] \\ 0 & x[4] & x[3] \\ 0 & 0 & x[4] \end{bmatrix}$$

$$R_{xx} = X^{*T}X$$

Covariance method.

$$X = \begin{bmatrix} x[2] & x[1] & x[0] \\ x[3] & x[2] & x[1] \\ x[4] & x[3] & x[2] \end{bmatrix}$$

$$R_{xx} = X^{*T}X$$

Modified-covariance method.

$$X = \begin{bmatrix} x[2] & x[1] & x[0] \\ x[3] & x[2] & x[1] \\ x[4] & x[3] & x[2] \end{bmatrix}$$

$$R_{xx} = X^{*T}X + \tilde{X}^T\tilde{X}^{*}$$

where \tilde{X} is the reversal of X .

References

- [1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 535-544, Prentice-Hall, 1992.

showeig

Purpose

Compute and display stem plot of eigenvalues.

Synopsis

`[h,e] = showeig(x,p)`

`[h,e] = showeig(x,p,'method')`

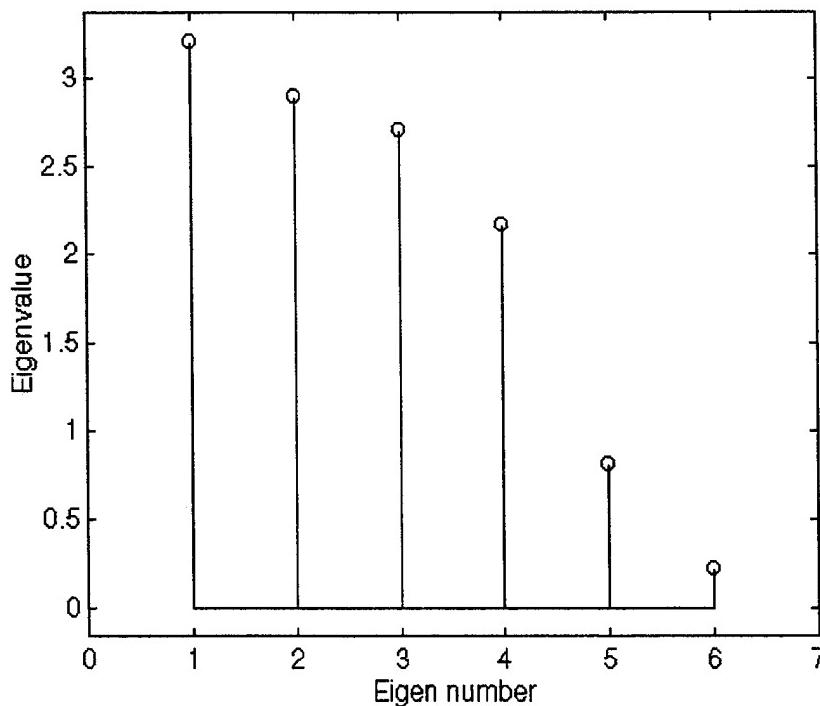
`[h,e] = showeig(X)`

Description

`[h,e] = showeig(x,p)` creates a stem plot of the eigenvalues computed from the $p \times p$ correlation matrix of the signal x . The correlation matrix is computed using the covariance method. A new figure window is always opened to display the plot and the eigenvalues are displayed in descending order from largest to smallest. The return h is a handle to the figure window the eigenvalues are displayed in. The eigenvalues are returned in vector e .

`[h,e] = showeig(x,p,'method')` used the specified 'method' in computing the estimation of the correlation matrix. Valid methods are 'rxxcorr', 'rxxcovar', or 'rxxmdcov'.

`[h,e] = showeig(X)` computes and displays the eigenvalues of the matrix X .



showeig

Examples

Create a random 3 x 3 matrix and display its eigenvalues.

```
X = randn(3,3);  
showeig(X);
```

Limitations

This function uses the SVD function to obtain eigenvalues and is thus subject to the same limitations as the SVD function.

Speech Function Reference

This section contains detailed descriptions of all speech signal processing functions in the Signal Processing and Communications Toolbox. It begins with a list of functions grouped by subject area and continues with the reference entries in alphabetical order. Information is also available through the online help facility.

Time-Domain Analysis

sp_steng	Short-time energy
sp_stmag	Short-time magnitude
sp_stzcr	Short-time zero-crossings

Curve smoothing

avsmooth	Average smoothing filter
mdsmooth	Median smoothing filter

Audio File Input/Output

loadau	Load Sun audio (*.au) file
saveau	Save vector to Sun Audio audio file
loadvoc	Load Soundblaster Creative Voice (*.voc) file
savevoc	Save vector to Soundblaster Creative Voice file
loadwav	Load Microsoft Windows wave (*.wav) file
savewave	Save vector to Microsoft Windows wave file

spcspch

avsmooth, mdsmooth

Purpose

Smooth a curve using an average or median smoothing filter.

Synopsis

```
y = avsmooth(x,L)
y = mdsmooth(x,L)
```

Description

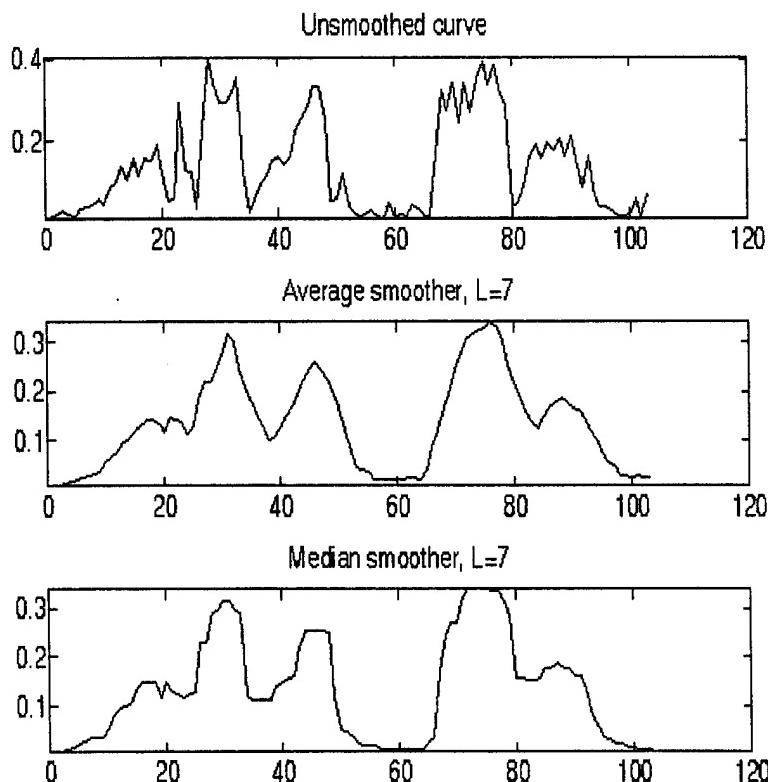
`avsmooth(x,L)` - Smooths the curve specified in vector x using an average smoothing filter of length L .

`mdsmooth(x,L)` - Smooths the curve specified in vector x using an median smoothing filter of length L .

Example

Compute the short-time energy of the speech signal specified in x and enhance the resulting curve using an average smoother of length 7.

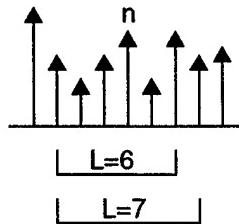
```
y = sp_steng(x,64,30,'hamming');
yy = avsmooth(y,7);
yyy = mdsmooth(y,7);
```



Algorithm

A non-causal smoothing algorithm is used. If L is odd, $\frac{L-1}{2}$ samples before and after

the current sample are used in computing the average or median. If L is even, $L/2$ samples before and $\frac{L}{2}-1$ samples after the current sample are used in computing the average or median. It is recommended an odd number for L be used to prevent any biasing an even number for L can cause.



Limitations

M-File Implementation

Zero padding is used on either end of the input vector and then the vector is trimmed to produce a non-causal filter. Therefore, samples within $L/2$ samples from either end will be subject to a transient which distorts the curve toward zero. This distortion becomes more exaggerated the further away the sample magnitudes in this region are from zero.

MEX-File Implementation

Zero padding used in the m-file implementation is not used, thus, the distortion towards zero does not occur. The mex-file version is considerably faster than the m-file version and is used automatically by MATLAB when both versions are located in the same directory.

See Also

[mdsmooth](#)

loadau, saveau

Purpose

Load and save data to and from audio files.

Synopsis

```
[y,fs] = loadau('filename')
saveau(x,'filename',fs)
```

Description

`[y,fs] = loadau('filename')` - Reads the audio (*.au) file in '*filename*' returning the audio data in *y* and the sampling frequency in *fs*. This function currently supports only 8-bit data. If the data is mu-law encoded, it is converted to linear values between plus and minus one. If '*filename*' does not have an extension, a ".au" extension is automatically added.

`saveau(x,'filename',fs)` - Converts *x* to mu-law encoded data and writes the data to the specified audio file setting the stored sampling rate to *fs*. If *fs* is not supplied, *fs* = 8000 Hz is used. If '*filename*' does not have an extension, an ".au" extension is automatically added.

Note: These functions are more versatile than the MATLAB functions `auread` and `auwrite` in that the sampling frequency is included.

Example

Load an audio file, cut the sampling rate in half and store to a new file:

```
[s,fs] = loadau('seatsit');
y = decimate(s,2);
saveau(y,'slowseat',y,fs/2);
```

See Also

`loadwav`, `savewav`, `loadvoc`, `savevoc`, `readsig`

loadvoc, savevoc

Purpose

Load and save data to and from Soundblaster Voice files.

Synopsis

```
[y,fs] = loadvoc('filename')
savevoc(x,'filename',fs)
```

Description

[y,fs] = loadvoc('filename') - Loads the data samples from a Soundblaster Creative Voice File (*.voc) into *y* and stores the sampling frequency into *fs*. The function checks the file to ensure it is a version 1.10 Soundblaster file. This function only supports raw, 8-bit, unpacked voice files. The filename extension ".voc" is appended to '*filename*' if no extension is supplied. Any mean is removed after being read from the voice file.

savevoc(*x*,*'filename'*,*fs*) - Saves the data in *x* to a Soundblaster Creative Voice File (*.voc) named '*filename.voc*'. Default sampling frequency is *fs* = 8192 Hz. This function saves the file as a version 1.10 file and stores the data in the raw, 8-bit unpacked format. The filename extension ".voc" is appended to '*filename*' if no extension is supplied.

Example

Load a voice file, cut the sampling rate in half and store to a new file:

```
[s,fs] = loadvoc('seatsit');
y = decimate(s,2);
savevoc(y,'slowseat',fs/2);
```

See Also

[loadwav](#), [savewav](#), [loadau](#), [saveau](#), [readsig](#)

Reference

[1] *The Developer Kit for Sound Blaster Series*, pp 4-6 to 4-12, Creative Labs, Inc., Nov 1991.

loadwav, savewav

Purpose

Load and save data to and from Microsoft Windows wave files.

Synopsis

```
[y,fs,bits,format] = loadwav('filename')
bits = savewav(x,'filename',fs)
bits = savewav(x,'filename',fs,bits)
```

Description

`[y,fs,bits,format] = loadwave('filename')` - Reads the file '*filename*' as a Microsoft Windows RIFF formatted *.wav audio file. If a filename extension is not included, a '.wav' extension is added. The function reads 8- and 16-bit PCM encoded wave files and returns the sampled data in *y*, sampling frequency in *fs* and number of bits-per-sample in *bits*. The return argument *format* is a 5x1 vector specifying

<i>format(1)</i>	format category (should be 1=PCM)
<i>format(2)</i>	number of channels
<i>format(3)</i>	sampling rate (same as fs)
<i>format(4)</i>	average bytes per second (for buffering)
<i>format(5)</i>	block alignment of data

Multi-channel data is supported and returned such that each column is a single channel.

Note: 8-bit integer data is stored as unsigned bytes in the range of 0 to 255. The `loadwav` function automatically subtracts 128 from the data so that it is centered about zero and will range from -128 to +127.

`savewav(x,'filename',fs)` - Writes the data in *x* to a Microsoft Windows wave (*.wav) formatted audio file. Data should be normalized for 8-bit (values 0 to 255) or 16-bit storage (values -32768 to +32767) before calling `savewav`. If 8-bit data is given in the range -128 to 127, an offset of 128 will be applied before storing the data. The function `loadwav` automatically removes this offset. If *x* is a matrix, the data will be stored as a multi-channel signal with data along the longest dimension considered a single channel. If the sampling frequency is not specified, it defaults to *fs* = 11.025 kHz.

`savewav(x,'filename',fs,bits)` - Forces the number of bits-per-sample to be *bits*. Valid values for *bits* are 8 or 16. The default bits-per-sample is dependent upon the data. If the data is in the range 0 to 255 or -128 to 127, the default is 8-bits. Data outside these ranges is considered 16-bit. An error message is produced if data is outside the range requested. The sampling frequency must be specified.

`savewav` returns the number of bits-per-sample the data was stored in.

loadwav

Example

Load a wave file, cut the sampling rate in half and store to a new file:

```
[s,fs] = loadwav('seatsit');
y = decimate(s,2);
y = 127 * (y / max(abs(y)));      % maximize dynamic range for 8-bit data
savewav(y,'slowseat',fs/2);
```

See Also

[loadvoc](#), [savevoc](#), [loadau](#), [saveau](#), [readsig](#)

readsig

Purpose

Read AU/VOC/WAV or flat integer signal files.

Synopsis

[y,fs] = readsig('filename')

[y,fs] = readsig('filename','bits')

Description

[y,fs] = readsig('filename.ext') - Reads the signal stored in the file 'filename.ext'. The file-name extension determines the load function used to read the file.

[y,fs] = readsig('filename',bits) - Reads the signal stored in a flat signed integer file format. The bits argument specifies the integer size in the number of bits. Giving a bits argument will also override the default read routine defined for the file extension. Support integer bit sizes are 8, 16, and 32.

Table 1: Support file types.

Extension	File type	Load function
*.au	Sun audio	loadau
*.voc	Creative Voice	loadvoc
*.wav	Microsoft Windows wave	loadwav
*.tim	SSPI time-domain signals	loadsspi
.	Flat integer files	ld8bit, ld16bit, ld32bit

Example

Load a Microsoft Windows wave file using readsig.

[y,fs] = readsig('seatsit.wav');

Limitations

If the file storage format does not include the sampling frequency, a value of fs = 1 is returned.

See Also

loadau, saveau, loadvoc, savevoc, loadwav, savewav, ld8bit, ld16bit, ld32bit, sv8bit, sv16bit, sv32bit

sp_steng, sp_stmag, sp_stzcr

Purpose

Compute the short-time energy, magnitude and zero-crossing curves of speech signals.

Synopsis

```
[y,t] = sp_steng(x,frame,overlap,fs,'window')
[y,t] = sp_stmag(x,frame,overlap,fs,'window')
[y,t] = sp_stzcr(x,frame,overlap,fs,'window')
```

Description

`sp_steng(x,frame,overlap,fs,'window')` - Computes the short-time energy of *x* using a window size of *frame* length (in milliseconds) and a percentage *overlap* between successive windows using the prefiltering data '*window*'. Available windows are '*rectangular*', '*hamming*', '*hanning*', '*blackman*', and '*bartlett*'. A rectangular window is the default if the '*window*' argument is not supplied. The output arguments are given by:

y - short-time function curve.

t - time indices for each corresponding to the start of each frame.

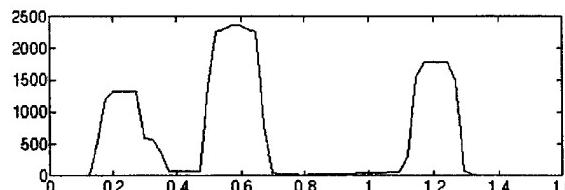
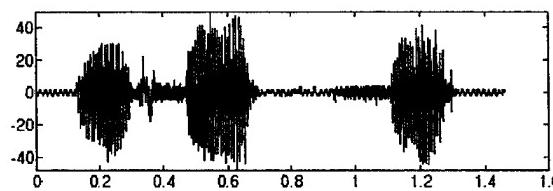
`sp_stmag(x,frame,overlap,fs,'window')` - Computes the short-time magnitude.

`sp_stzcr(x,frame,overlap,fs)` - Computes the short-time zero-crossing rate.

Examples

Compute and display the short-time energy using a 30 millisecond frame, a 20% frame overlap, a Hamming window and a median smoothing filter. Note the steps necessary to plot the graphs on the same scale and aligned in time.

```
[y,t] = sp_steng(x,0.03,20,fs,'hammin');
yy = mdsmooth(y,7);
subplot(2,1,1);plot((0:length(x)-1)/fs,x);
subplot(2,1,1);plot(t,yy)
```



Algorithm

Short-time energy:

$$E_n = \sum_{m=-\infty}^{\infty} |[x(m) w(n-m)]|^2$$

Short-time magnitude:

$$M_n = \sum_{m=-\infty}^{\infty} |x(m)|w(n-m)$$

Short-time zero crossings:

$$Z_n = \sum_{m=-\infty}^{\infty} |\operatorname{sgn}[x(m)] - \operatorname{sgn}[x(m-1)]|w(n-m)$$

Note: Both m-file and mex-file implementations are supplied. The mex-file version will take precedence over the m-file version when both the m-file and mex-file are located in the same directory.

See Also

`avsmooth`, `mdsmooth`

Reference

L. R. Rabiner & R. W. Schafer, *Digital Processing of Speech Signals*, pp 120-130, Prentice Hall, 1978.

sp_steng

SSPI Function Reference

This section contains detailed descriptions of all SSPI related functions in the Signal Processing and Communications Toolbox. It begins with a list of functions grouped by subject area and continues with the reference entries in alphabetical order. Information is also available through the online help facility.

SSPI Functions	
loadsspi	Load SSPI formatted signal file
savesspi	Save vector to SSPI formatted signal file
loadsurf	Load cyclostationary surface produced by SXAF See help loadsurf
sspipam	SSPI PAM program interface See help sspipam
sspisxaf	SSPI SXAF program interface See help sspisxaf

spcsspi

loadsspi, savesspi

Purpose

Load data stored in the format specified by the Cyclic Spectral Analysis Software Package from Statistical Signal Processing, Inc.

Synopsis

```
y = loadsspi('name');  
y = loadsspi('name','binary');  
savesspi(x,'name');
```

Description

`loadsspi('name')` - Loads SSPI software generated data from the file *name*. Data is stored in ASCII format.

`loadsspi('name','binary')` - Data is stored in binary format.

`savesspi(x,'name')` - Saves the vector *x* to the file *name* in SSPI data file format. Data is written in floating point ASCII.

`savesspi(x,'name','binary')` - Data is stored in binary format.

ASCII SSPI format is:

first line - *type nbrsamples*

rest of lines - one or two columns of ascii numeric data

where,

type: 1 = real, 2 = complex

nbrsamples = number of samples (one sample per line).

```
2 6  
3.226078e-01 2.978590e+00  
5.433282e-01 8.382938e-01  
3.492872e+00 9.382934e+00  
3.293842e-01 2.394829e+00  
4.592839e+01 2.293940e+00  
6.203482e+00 2.382920e+01
```

Binary format follows the same line except data is store serially.

Reference

- [1] Stephan V. Schell and Chad M. Spooner, Cyclic Spectral Analysis Software Package User's Manual, Statistical Signal Processing, Inc., 1991.

loadsspi

Utility Function Reference

This section contains detailed descriptions of all utility functions in the Signal Processing and Communications Toolbox. It begins with a list of functions grouped by subject area and continues with the reference entries in alphabetical order. Information is also available through the online help facility.

Data File Input/Output

ld8bit,sv8bit	Load/save signed 8-bit data
ld16bit, sv16bit	Load/save signed 16-bit data
ld32bit, sv32bit	Load/save signed 32-bit data

Plotting Functions

plottime	Plot time-domain signal
lperigrm	Display FFT on logarithmic scale
wperigrm	Display FFT on linear scale
xpilabel	Label X-axis in units of π .
ypilabel	Label Y-axis in units of π .

Data Functions

framdata	Form matrix containing overlapping frames
-----------------	---

Miscellaneous

fedit	Edit function M-file source or MEX-file source
--------------	--

spcutil

fedit

Purpose

Edit function M-file source or MEX-file source.

Synopsis

fedit function

Description

fedit function opens a text editor and loads the source for the m-file function. If function is a MEX file, the C or FORTRAN source file is opened if it resides in the same directory as the MEX file. If function is a built-in function or a variable or is not found, an appropriate error message is produced.

Examples

Edit the fedit function to customize the editor used under MS-Windows MATLAB to be the MS-Windows Notepad editor.

fedit fedit

Note: This example will not work if your current platform is MS-Windows and you are not using Norton Desktop (or the directory *ndw* is not in your path). To make the changes described below, you will have to manually load the file ..\spc-tools\spcprog\fedit.m into your favorite editor.

Find the following section of code

```
% form the command
if strcmp(computer,'SUN4'),
    cmd = ['!textedit -Wt cour.r.14 'f' &'];
elseif strcmp(computer,'PCWIN'),
    cmd = ['!deskedit 'f' &'];
else
    error('fedit: An editor for this computer has not been defined. See FEDIT.')
end;
```

and change "deskedit" to "notepad" in the line below the line containing 'PCWIN'.

To add support for another platform, add the following lines above the "else" line

```
elseif strcmp(computer,'string'),
    cmd = ['!editor 'f' &'];
```

where 'string' is that returned by the computer command and "editor" is the program name of the editor of your choice.

See Also

computer

framdata

Purpose

Form overlapping frames of vectored data.

Synopsis

[y,tscale] = framdata(x,Nframe,Ncolumn,Noverlap,fs)

Description

framdata(x,Nframe,Ncolumn,Noverlap) - Forms a matrix whose columns contain individual frames of data vector *x*. Each column of the output matrix *y* is formed from *Nframe* consecutive samples zero padded out to *Ncolumn* samples-per-column (if necessary). Each consecutive column is formed by taking the last *Noverlap* samples from the previous frame as its first portion.

Nframe and *Ncolumn* must meet the condition

$$0 < Nframe \leq Ncolumn$$

Nframe and *Noverlap* must meet the condition

$$0 \leq Noverlap < Nframe$$

[y,tscale] = framdata(x,Nframe,Ncolumn,Noverlap,fs) - Returns the output vector *tscale* whose values correspond to the start times of each frame in *y* based on the sampling frequency *fs*. The sampling frequency defaults to *fs* = 1 if not given.

Note: The last frame receives additional zero padding if there are not enough elements remaining in *x* to fill the frame with *Nframe* samples.

Example

Form a matrix where each column contains three samples and one zero-pad with a one-sample overlap between frames. Obtain a vector containing the frame start times based on a sampling frequency of 2 Hz.

```
>> x = 1:10
x =
    1   2   3   4   5   6   7   8   9   10

>> [y,t] = framdata(x,3,4,1,2)
y =
    1   3   5   7   9
    2   4   6   8   10
    3   5   7   9   0
    0   0   0   0   0
t =
    0
    1
    2
    3
    4
```

ld8bit, ld16bit, ld32bit, sv8bit, sv16bit, sv32bit

Purpose

Load/save flat signed integer data files.

Synopsis

```
y = ld8bit('filename');
y = ld8bit('filename',number)
y = ld8bit('filename',number,offset)
y = ld8bit('filename',0,offset);

error = sv8bit('filename',x)
```

Description

`ld8bit('filename')` - Loads the entire file '*filename*' as 8-bit signed samples.

`ld8bit('filename',number)` - Loads the first *number* of samples starting from the beginning of the file.

`ld8bit('filename',number,offset)` - Loads *number* of samples beginning at *offset* from the beginning of the file. If *number* = 0, loading starts from *offset* and continues to the end-of-file.

The `ld16bit` and `ld32bit` commands function identically to the `ld8bit` command.

`sv8bit('filename',x)` - Saves the data in vector *x* to the file '*filename*' as 8-bit signed integers. Data outside the range of 8-bits is clipped and a warning message is printed.
`sv8bit` returns *error* = 1 failure, *error* = 0 success.

The `sv16bit` and `sv32bit` commands function identically to the `sv8bit` command.

Example

The file *datafile.dat* contain four seconds of data sampled at 8192 Hz. Load the data collected during the third second.

```
y = ld8bit('datafile.dat',2*8192+1,8192);
```

(The value “2*8192+1” references the data sample immediately after the data from the first two seconds as the starting point.)

lperigrm

Purpose

Display the periodogram of a signal on a logarithmic scale.

Synopsis

lperigrm(x)
lperigrm(x,fs)
lperigrm(x,mindb)
lperigrm(x,fs,mindb)
lperigrm(x,fs,'phase')
lperigrm(x,fs,mindb,'phase')

Description

lperigrm(x) - Computes the periodogram of *x* and displays the result on a logarithmic (decibel) scale. Only the positive frequencies are displayed. The default sampling frequency is *fs* = 8192 Hz.

lperigrm(x,fs) - Sets the sampling frequency to *fs*.

lperigrm(x,mindb) - Cuts the plot off below *mindb* for a better display of the data above. If *mindb* is positive, it is interpreted as *fs* in the above usage. Therefore, *mindb* must be negative for this usage.

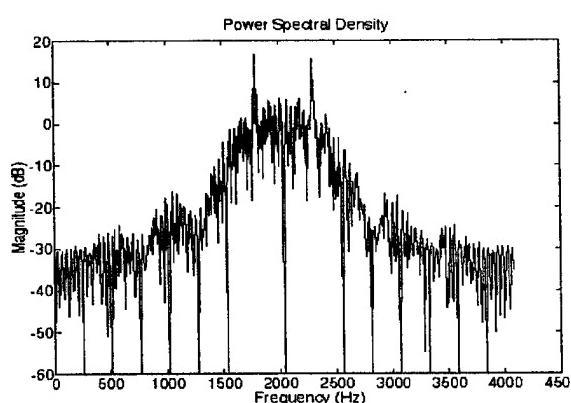
lperigrm(x,fs,mindb) - Used to set *fs* and *mindb* to other than their default values or to set *mindb* when *mindb* is positive.

lperigrm(x,fs,'phase') and *lperigrm(x,fs,mindb,'phase')* - Splits the graphics window and displays both the magnitude and phase information.

Example

Generate a frequency-shift keyed signal and display its periodogram. Display the magnitude down to -100 dB.

```
y = bpsk(512,512,2048,0.1);
lperigrm(y,-100);
```



Algorithm

The periodogram is given by:

$$S_{x_T}(f) \equiv \frac{1}{T} |X_T(f)|^2$$

where $X_T(f)$ is the fourier transform of $x(t)$

$$X_T(f) = \int x(u) e^{-j2\pi f u} du$$

T is the frame length and $x(u)$ is

$$x(u) = x_T(u)w(u)$$

where $w(u)$ is the data smoothing window.

The plot magnitude is

$$20 \log (|X_T(f)|)$$

Limitations

The size of the FFT used to compute the Fourier transform is the closest power-of-two greater-than or equal-to the length of x . Long input vectors require longer computation times. If x is short and a finer frequency resolution is desired, the input vector x can be zero padded before calling the function.

`lperigrm([x ; zeros(300,1)]);`

See Also

`wperigrm`, `plottime`

Reference

- [1] William A. Gardner, *Statistical Spectral Analysis, A Nonprobabilistic Theory*, pp. 5-7, Prentice-Hall, 1988.

plottime

Purpose

Plot the time-domain display of a signal.

Synopsis

```
plottime(x)
plottime(x,fs)
plottime(x,duration)
plottime(x,fs,duration)
plottime(x,duration,offset)
plottime(x,fs,duration,offset)
```

Description

`plottime(x)` - Plots the time-domain representation of *x* with a time axis scaled for a sampling frequency of 8192 Hz. Default *duration* is one second.

`plottime(x,fs)` - Sets the sampling frequency to *fs* Hz. The variable *fs* must be greater than or equal to 100 Hz for this usage.

`plottime(x,duration)` - Plot up to *duration* seconds of *x*. The *duration* must be less than 100 seconds for this usage.

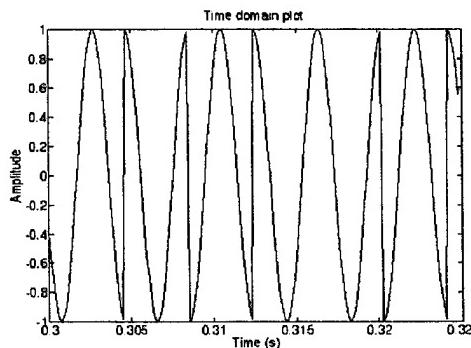
`plottime(x,fs,duration)` - Plots up to *duration* seconds of *x*.

`plottime(x,duration,offset)` and `plottime(x,fs,duration,offset)` - Starts the plot *offset* seconds from zero seconds.

Example

Generate one second of a binary phase-shift signal. Display 1/40 seconds of the signal beginning at 0.3 seconds.

```
y = bpsk(256,256);
plottime(y,1/40,3);
```



See Also

`lperigrm`, `wperigrm`

wperigrm

Purpose

Display the periodogram of a signal using a linear scale (relative magnitude).

Synopsis

```
wperigrm(x)
wperigrm(x,fs)
wperigrm(x,fs,'phase')
```

Description

`wperigrm(x)` - Computes the periodogram of x and displays the result using a linear (relative magnitude) scale. Only the positive frequencies up to half the sampling frequency are displayed. The default sampling frequency is 8192 Hz.

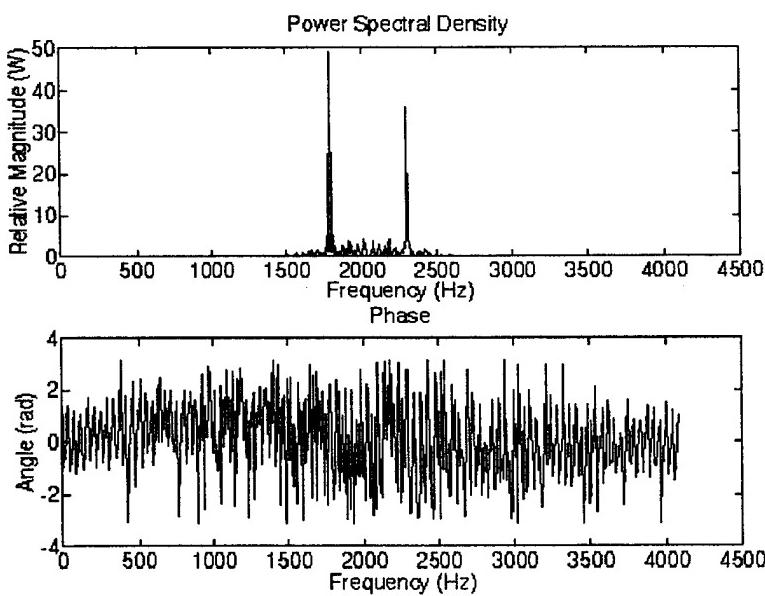
`wperigrm(x,fs)` - Sets the sampling frequency to fs .

`wperigrm(x,fs,'phase')` - Splits the graphics window and displays both the magnitude and phase information.

Example

Generate a frequency-shift keyed signal and display its periodogram.

```
y = bpsk(512,512,2048,0.1);
wperigrm(y,'phase');
```



Algorithm

The periodogram is given by:

$$S_{x_T}(f) \equiv \frac{1}{T} |X_T(f)|^2$$

where $X_T(f)$ is the fourier transform of $x(t)$

$$X_T(f) = \int x(u) e^{-j2\pi f u} du$$

T is the frame length and $x(u)$ is

$$x(u) = x_T(u) w(u)$$

where $w(u)$ is the data smoothing window.

Limitations

The size of the FFT used to compute the Fourier transform is the closest power-of-two greater-than or equal-to the length of x . Long input vectors require longer computation times. If x is short and a finer frequency resolution is desired, the input vector x can be zero padded before calling the function.

```
wperigrm([x ; zeros(300,1)]);
```

See Also

lperigrm, plottime

Reference

- [1] William A. Gardner, *Statistical Spectral Analysis, A Nonprobabilistic Theory*, pp. 5-7, Prentice-Hall, 1988.

xpilabel, ypilabel

Purpose

Label X- or Y-axis in units of π .

Synopsis

`xpilabel`

`ypilabel`

`xpilabel(h)`

`ypilabel(h)`

`xpilabel('text')`

`ypilabel('text')`

`xpilabel(h,'text')`

`ypilabel(h,'text')`

Description

`xpilabel` changes the X-axis tick marks to multiples or fractions of π and labels tick marks accordingly in the form "3 π " or "3 $\pi/2$ ".

`xpilabel(h)` changes the X-axis tick marks in the axes pointed to by the axes handle `h`.

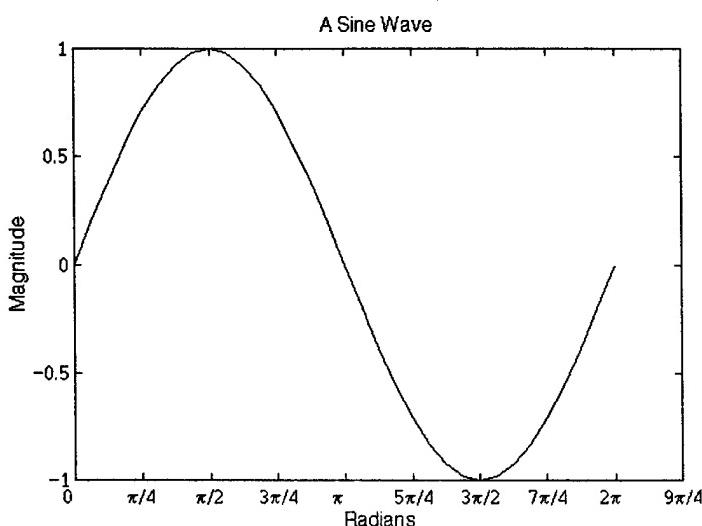
`xpilabel('text')` and `xpilabel(h,'text')` also labels the X-axis with 'text' and can be used in place of the `xlabel` command.

Note: Because the axis label text and title text use the same font used in the `axes` object, the `ylabel`, `xlabel`, or `title` commands must be executed before `xpilabel` and `ypilabel` to avoid the axis labels or title from using the Symbol font. These functions change the axes font to the Symbol font in order to display the Greek π character.

Examples

Plot one cycle of a sine wave and label the X-axis in radians.

```
t = 0:pi/16:2*pi;
y = sin(t);
plot(t,y);
title('A Sine Wave');
ylabel('Magnitude');
xpilabel('Radians');
```



xpilabel

Programming Function Reference

This section contains detailed descriptions of all programming functions in the Signal Processing and Communications Toolbox. It begins with a list of functions grouped by subject area and continues with the reference entries in alphabetical order. Information is also available through the online help facility.

Axes Cursors	
crsrmake	Create horizontal or vertical cursor object
crsrmove	Move cursor
crsrlloc	Return current cursor location
crsrlind	Return current indices into attached line object
crsron	Make cursor visible
crsroff	Make cursor invisible
crsrdel	Delete cursor from axes
crsrdown	Button down callback function
crsrmov	Button down/cursor drag callback function
crsrup	Button up callback function

Object Handle Retrieval	
findaxes	Find axes object (UserData)
findchkb	Find check box uicontrol (String, String & UserData)
findedit	Find edit box uicontrol (UserData, Value)
findfig	Find figure object (Name, Name & UserData)
findfram	Find frame uicontrol (UserData)
findline	Find line object (UserData)
findmenu	Find uimenu (Label [[& Label] & Label])
findpopu	Find popup menu uicontrol (UserData)
findpush	Find push button uicontrol (String, String & UserData)

Object Handle Retrieval	
findradio	Find radio button uicontrol (String, String & UserData)
findslid	Find slider uicontrol (String, String & UserData)
findtext	Find axes text object (String, String & UserData)
finduitx	Find text uicontrol (String, String & UserData)

Menu Objects	
workmenu	Add “Workspace” menu and callback structure
mufirwin	Add FIR window menu
gtfirwin	Get FIR window vector based on FIR window menu
menucmap	Add Colormap checked menu
menushad	Add Shading checked menu
uncheckm	Uncheck uimenu children

Popup Menu Objects	
popunbrs	Create numerical popup menu and callback structure
callnbrs	Callback function for popunbrs
getpopvl	Get current numeric value from popup menu
getpopst	Get current string value from popup menu

ZoomTool	
zoomtool	Add zoomtool to axes
zoomloc	Return zoomtool cursor locations
zoomind	Return zoomtool cursor indices in attached line
zoomset	Move zoomtool cursors
zoomclr	Remove zoomtool from axes
zoomdown	Button down callback function
zoommove	Button down/cursor drag callback function
zoomup	Button up callback function
zoomleft	Move cursor left one sample
zoomright	Move cursor right one sample
zoompklf	Move cursor to next extrema on left
zoompkrt	Move cursor to next extrema on right
zoomxin	Zoom X-axis in
zoomxout	Zoom X-axis out
zoomxful	Zoom X-axis to full limits
zoomyin	Zoom Y-axis in
zoomyout	Zoom Y-axis out
zoomyful	Zoom Y-axis to full limits
zoomtggl	Toggle cursor between multiple lines
zoomline	Replace line object with cursors attached
zoomplay	Add play menu and add sound capabilities
zoomedit	Add edit menu and edit capabilities
zoomedsv	Save callback script for edit menu

Strings	
dblquote	Replace single quotes with two single quotes
reqstr	Request string input dialog box and callback structure.

Strings

ednbrchk	Check edit box string for numeric validity
-----------------	--

Miscellaneous

snapshot	Copy axes to new figure window
cntaxes	Count/return handles of axes objects
cntlines	Count/return handles of axes line objects
cntuictrl	Count/return handles of uicontrol objects

crsrmake, crsrmov, crsrlc, crsron, crsroff, crsrdel

Purpose

Axis cursor object manipulation.

Synopsis

```

h = crsrmake(axes,'name','direction',position,'linetype')
h = crsrmake(axes,'name','direction',position,'linetype','callback')
h = crsrmov(axes,'name',position)
y = crsrlc(axes,'name')
h = crsron(axes,'name')
h = crsroff(axes,'name')
h = crsrdel(axes,'name')

```

Description

The crsr family of functions provides high-level routines for creating and manipulating axis cursor objects. An axis cursor is a line object that can be manipulated with the crsr family of functions or the mouse. Cursor objects can serve as cursors on axis displays that emulate cursors on equipment such as digital storage oscilloscopes and spectrum analyzers. See zoomtool.m for an example of a program which makes use of cursor objects.

`crsrmake(axes,'name','direction',position,'linetype')` creates a cursor object on the axis pointed to by the handle `axes`. The 'name' argument is a unique identifier for the cursor and is used to reference the cursor during subsequent cursor operations. The 'name' argument can be either a string or a scalar number. The orientation 'direction' argument can be either 'horizontal' or 'vertical'. The position argument is a scalar number for the location to place the cursor (x-axis location for 'vertical' cursors, y-axis location for 'horizontal' cursors). The 'linetype' argument is any valid MATLAB linetype. An error will be generated if an attempt to create a cursor beyond the axis limits occurs.

`crsrmake(axes,'name','direction',position,'linetype','callback')` enables "dragging" the cursor object with the mouse pointer. After the cursor has been moved with the mouse or the `crsrmov` function, the 'callback' string is evaluated. The callback string can be any legal MATLAB expression, including the name of any M-file or function. A null callback string will allow the cursor object to be dragged without any action taking place after cursor movement. When cursor dragging is enabled, the user can grab a cursor by positioning the mouse pointer over the cursor and pressing the left mouse button. While holding the left mouse button down, the user can then move the cursor within the axes limits. For vertical cursors, only the horizontal position of the mouse pointer is of importance. For horizontal cursors, only the vertical position of the mouse pointer is of importance.

`crsrmov(axes,'name',position)` moves the named cursor to a new location. It does not change the visibility of the cursor. Cursor movement beyond the current axes limits is permitted.

`crsrlc(axes,'name')` returns the x- or y-axis cursor location dependent upon if the cursor is a vertical or horizontal cursor.

`crsron(axes,'name')` turns the named cursor visibility on.

`crsroff(axes,'name')` turns the named cursor visibility off. Turning a cursor's visibility off does not disable using the `crsrmove`, `crsrlc`, or any other `crsr` family functions on the named cursor. It does hide the cursor from the user and since it is hidden, the user cannot grab the cursor with the mouse and move it. An example of where this capability is useful is in the `zoomtool.m` program. Setting the `HorizontalCursors` property to `off` in `zoomtool` actually just hides the horizontal cursors from the user's view.

`crsrdel(axes,'name')` deletes the named cursor object.

Examples

Create a cursor object, play with it for a while and then delete it.

```
figure; gca;
crsrmake(gca,'cursor','horizontal',0.5,:);
crsrmove(gca,'cursor',0.3);
crsroff(gca,'cursor');
crsrmove(gca,'cursor',0.7);
crsron(gca,'cursor');
crsrdel(gca,'cursor');
```

Related Files

- `crsrdown.m` - Mouse button down callback function.
- `crsrmmov.m` - Mouse button down/cursor movement callback function.
- `crsrup.m` - Mouse button up callback function.

findaxes

Purpose

Find an axes graphic object.

Synopsis

```
h = findaxes(figure,'identifier')
h = findaxes(figure,identifier)
```

Description

*findaxes(*figure*,*'identifier'*)* finds the axes graphic object with the *UserData* property equal to the string '*identifier*' in the figure window specified with the handle *figure*. The '*identifier*' and *Userdata* property can be either a string or a scalar number and must match exactly.

Examples

Create an axes object and assign it the identifier '*magplot*'.

```
subplot(2,1,1);
set(gca,'UserData','magplot');
```

Retrieve the handle to the 2,1,1 subplot.

```
h = findaxes(gcf,'magplot');
```

See Also

findchkb, *findedit*, *findfig*, *findfram*, *findline*, *findmenu*, *findpopu*, *findpush*, *findrdo*, *findslid*, *findtext*, *finduitx*

findchkb, findpush, findrrio, findslid, finduitx

Purpose

Find a check box , push button, radio button, slider, or text style uicontrol.

Synopsis

```
h = findchkb(figure,'label')
h = findchkb(figure,'label','identifier')

h = findpush(figure,'label')
h = findpush(figure,'label','identifier')

h = findrrio(figure,'label')
h = findrrio(figure,'label','identifier')

h = findslid(figure,'label')
h = findslid(figure,'label','identifier')

h = finduitx(figure,'label')
h = finduitx(figure,'label','identifier')
```

Description

`findchkb(figure,'label')` finds the check box style uicontrol with the `String` property equal to '`label`' in the figure window specified with the handle `figure`. Both '`label`' and the `String` property must be strings and match exactly.

`findchkb(figure,'label','identifier')` finds the check box style uicontrol with the `String` property equal to '`label`' and with the `Userdata` property equal to '`identifier`'. The '`identifier`' and `Userdata` property can be either a string or a scalar number and must match exactly. This use is convenient when a figure window contains two or more check box controls with the same label.

Examples

Create three check box style uicontrols.

```
uicontrol(gcf,'Style','Checkbox','String','X','Userdata',1,...);
uicontrol(gcf,'Style','Checkbox','String','X','Userdata','second',...
uicontrol(gcf,'Style','Checkbox','String','O',...);
```

Retrieve the handle to the second check box labeled 'X'.

```
h = findpush(gcf,'X','second');
```

Retrieve the handle to the check box labeled 'O'.

```
h = findpush(gcf,'O');
```

See Also

`findaxes`, `findedit`, `findfig`, `findfram`, `findline`, `findmenu`, `findpopu`, `findtext`

findedit, findpopu

Purpose

Find an edit box or popup menu style uicontrol.

Synopsis

```
h = findedit.figure,'identifier')
h = findedit.figure,identifier)
```

```
h = findpopu.figure,'identifier')
h = findpopu.figure,identifier)
```

Description

`findedit.figure,'identifier')` finds the edit box style uicontrol with the `UserData` property equal to '`identifier`' in the figure window specified with the handle `figure`. The '`identifier`' and `Userdata` property can be either a string or a scalar number and must match exactly.

Examples

Create two edit uicontrols.

```
uicontrol('Style','Edit','UserData',1,...);
uicontrol('Style','Edit','UserData','second',...);
```

Retrieve the handle to the second edit uicontrol.

```
h = findpush(gcf,'second');
```

See Also

`findaxes`, `findchkb`, `findfig`, `findfram`, `findline`, `findmenu`, `findpush`, `findrdio`,
`findslid`, `findtext`, `finduitx`

findfig

Purpose

Find a figure window.

Synopsis

```
h = findfig('figure','name')
h = findfig('figure','name','identifier')
```

Description

`h= findfig('name')` finds the *figure* window with the Name property equal to the string '*name*'. The Name property determines the text displayed in the window titlebar. Both '*name*' and the Name property must be a string and must match exactly.

`h = findfig('name','identifier')` finds the *figure* window with the Name property equal to the string '*name*' and with the Userdata property equal to '*identifier*'. The '*identifier*' and Userdata property can be either a string or a scalar number and must match exactly.

Examples

Create a figure and assign it the name '*My Application*'.

```
figure;
set(gca,'Name','My Application');
```

Retrieve the handle to the figure window.

```
h = findfig(gcf,'My Application');
```

See Also

`findaxes`, `findchkb`, `findedit`, `findfram`, `findline`, `findmenu`, `findpopu`, `findpush`, `findrdio`, `findslid`, `findtext`, `finduitx`

findfram

Purpose

Find a frame style uicontrol.

Synopsis

`h = findfram(figure,'identifier')`

Description

`findfram(figure,'identifier')` finds the frame style uicontrol with the String property equal to the string '*identifier*'. Both '*identifier*' and the String property must be strings and match exactly.

Examples

Create two frame style uicontrols.

```
uicontrol('Style','Frame','UserData','first',...
uicontrol('Style','Frame','UserData','second');
```

Retrieve the handle to the second frame.

```
h = findfram(gcf,'second');
```

See Also

`findaxes`, `findchkb`, `findedit`, `findfig`, `findline`, `findmenu`, `findpopu`, `findrdio`,
`findslid`, `findtext`, `finduitx`

findline

Purpose

Find an axes line object.

Synopsis

```
h = findline(axes,'identifier')
h = findline(axes,identifier)
```

Description

`findline(axes,'identifier')` finds the axes line object with the `UserData` property equal to `'identifier'` in the axis specified with the handle `axes`. The `'identifier'` and `Userdata` property can be either a string or a scalar number and must match exactly.

Examples

Create a line and assign it the identifier `'myline'`.

```
line([0 1],[0 1],'UserData','myline');
```

Change its color to red

```
set(findline(gca,'myline'),'Color','red');
```

See Also

`findchkb`, `findedit`, `findfig`, `findfram`, `findmenu`, `findpopu`, `findpush`, `findrdio`,
`findslid`, `findtext`, `finduitx`

findmenu

Purpose

Find a uimenu object.

Synopsis

```
h = findmenu('figure','menu')
h = findmenu('figure','menu','submenu')
h = findmenu('figure','menu','submenu','submenu')
```

Description

`findmenu('figure','menu','submenu','submenu')` searches the uimenu object tree to find the uimenu object with the Label property equal to the last '*menu*' or '*submenu*' argument. If no two submenus in the pull-down uimenu structure are the same, the '*menu*' and any higher level '*submenu*'s can be left off. Both '*name*' or '*submenu*' and the Label property must be strings and match exactly.

Examples

Draw the menu.

```
Workspace
  New Figure
  Save
    As
    Was
  Exit
```

```
f = uimenu('Label','Workspace');
uimenu(f,'Label','New Figure');
f2 = uimenu(f,'Label','Save');
uimenu(f2,'Label','As');
uimenu(f2,'Label','Was');
uimenu(f,'Label','Exit');
```

Change the second-level submenu "Was" to "Were".

```
set(findmenu(gcf,'Workspace','Save','Was'),'Label','Were')
```

See Also

`findaxes`, `findchkb`, `findedit`, `findfig`, `findfram`, `fineline`, `findpopu`, `findpush`, `findrdio`,
`findslid`, `findtext`, `finduitx`

findtext

Purpose

Find an axes `text` object.

Synopsis

```
h = findtext(axes,'text')
h = findtext(axes,'text','identifier')
```

Description

`findtext(figure,'text')` finds the axes `text` object with the `String` property equal to '`text`' in the axis specified with the handle `axes`. Both '`label`' and the `String` property must be strings and match exactly.

`findtext(figure,'text','identifier')` finds the axes `text` object with the `String` property equal to '`text`' and with the `Userdata` property equal to '`identifier`'. The '`identifier`' and `Userdata` property can be either a string or a scalar number and must match exactly. This use is convenient when a figure window contains two or more check box controls with the same label.

Examples

Create a `text` object and assign it the identifier '`mytext`'.

```
text(0.5,0.4,'mytext');
```

Change its color to red

```
set(findtext(gca,'mytext'),'Color','red');
```

See Also

`findchkb`, `findedit`, `findfig`, `findfram`, `findmenu`, `findpopu`, `findpush`, `findrdo`,
`findslid`, `findtext`, `finduitx`

mufirwin, gtfirwin

Purpose

FIR filter window menu.

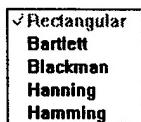
Synopsis

`h = mufirwin(figure,'callback')`

`y = gtfirwin(figure,N)`

Description

`h = mufirwin(figure,'callback')` adds a menu to the figure window pointed to by the handle, `figure`, which allows the user to select a pre-defined FIR window type. The argument '`callback`' is an optional string which defines a common CallBack function for all the submenu items. The menu is labeled Window on the figure menu bar.



The Hanning window is the default window. The current window is always the one containing the checkmark in the menu. The Signal Processing Toolbox is required for this function.

`y = gtfirwin(figure,N)` returns a `N` length FIR window according to the current FIR window selected in the Window menu in the figure pointed to by the handle `figure`.

Examples

Create a FIR window menu.

```
mufirwin(gf,'sigical("drawtrans");');
```

Create an FIR window based on the setting of the FFT popup menu and FIR window settings.

```
fft_length = getpopvl(gf,'FFT')
window = gtfirwin(gf,fft_length)
```

radiogrp

Purpose

Create radio button group.

Synopsis

`h = radiogrp(figure,'label','items',pushed,llcorner,button,'property','propertyvalue',...)`

Description

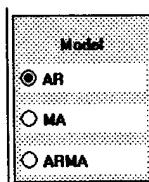
`h = radiogrp(figure,'label','items',pushed,llcorner,button,'property','propertyvalue',...)` creates a radio group named '*label*' containing buttons named '*items*' in the figure specified by the handle *figure*. The argument *pushed* is the row number of the radio button to push. *LLCORNER* is a two element vector [left bottom] specifying the lower left corner of the frame surrounding the radio group. *BUTTON* is a four-element vector [width height frame interval] specifying the button size.

Properties:

CallBack	<str>
Units	pixels 'normal' inches centimeters points
ForeGroundColor	colorspec
BackGroundColor	colorspec

Examples

Create the following radio button group.



```
items = str2mat('AR','MA','ARMA');
radiogrp(gf,'Model',items,1,[10 20],[100 22 6 10],'Units','pixels');
```

togmenu, getcheck, togcheck

Purpose

Create menu with toggling checkmarked menu items.

Synopsis

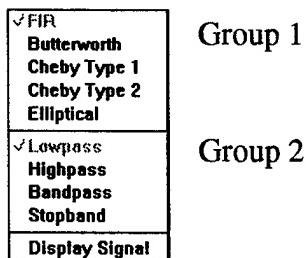
```

h = togmenu('figure','label','items')
h = togmenu('figure','label','items',check)
h = togmenu('figure','label','items',check,callback)
h = getcheck('figure','label')
h = getcheck('figure','label',group)
toggcheck

```

Description

The **togmenu** command creates groups of menu items which are automatically toggled when the user selects an unchecked menu item in a group. For example, the **togmenu** command is used to create the **Filter** menu in the **sigfilt** tool containing two groups of togglable menu items.



Selecting **Cheby Type 2** in this menu will automatically uncheck and enable the **FIR** menu item, check and disable the **Cheby Type 2** menu item and then call a common **CallBack** function for Group 1. Likewise, selecting **Highpass** will automatically uncheck and enable the **Lowpass** menu item, check and disable the **Highpass** menu item and then call a common **CallBack** function for Group 2. Note that checked menu items are disabled to prevent the user from selecting a previously checked menu item (thereby implementing the **CallBack** function twice). See the example below for more information on setting up and using togglable menu item groups.

h = togmenu('figure','label','items') creates a pull-down menu called '**label**' with submenu items specified in the string matrix '**items**' in the figure window specified by the handle **figure**. The return argument **h** is the handle to the parent **uimenu** object. By default, the first string in row one of '**items**' is checked when the corresponding menu item is created. Calling **togmenu** multiple times with different '**items**' will create separated groups of togglable menu items under the same menu label (see example below).

h = togmenu('figure','label','items',check) checks the menu item specified in row **check** in the '**items**' string matrix and sets up the menu for use with **getcheck** to retrieve the handle to the checked menu item at run-time.

h = togmenu('figure','label','items',check,callback) assigns the '**callback**' function specified in

the string 'callback' to each menu item.

h = getcheck(figure,'label') returns the handle of the checked uimenu object child on the menu 'label' in the figure window specified with the handle figure. To use getcheck, the menu has to be created using the togmenu command. By default, the togmenu group is group one.

h = getcheck(figure,'label',group) returns the handle of the checked uimenu child in the group number of grouped toggable menu items as setup by the togmenu command.

togcheck(group) toggles the checkmarks in a group of pull-down uimenu submenu items created with the togmenu command. This function is automatically placed at the beginning of the common CallBack function for groups created with the togmenu command. In general, users and programmers have no need to call this function directly.

Examples

The following code initializes the Filter menu in the sigedit tool

```
% Filter menu

% Filter prototypes will be group 1.
items = str2mat('FIR','Butterworth','Cheby Type 1','Cheby Type 2','Elliptical');
togmenu(gf,'Filter',items,1,'sigfical("setfilter")');

% Filter types will be group 2.
items = str2mat('Lowpass','Highpass','Bandpass','Stopband');
togmenu(gf,'Filter',items,1,'sigfical("settype")');

% Also add a menu item to display the time domain signal.
uimenu(findmenu(gf,'Filter'),'Label','Display Signal',...
    'Separator','on','CallBack','sigfical("killtime")');
```

When the user selects a new filter prototype in group 1, the 'setfilter' code section of *sigfical.m* determines which filter prototype is checked with the code

```
vf_filter = lower(get(getcheck(gf,'Filter',1),'Label'));
```

In this line, the "getcheck(gf,'Filter',1)" code returns the handle of the checked menu item in group 1, the filter prototype group, which the get function then used to retrieve the checked menu items uimenu Label property. Additional code in the 'setfilter' section of *sigfical.m* use code such as

```
if strcmp(vf_filter,'fir'),
    % Code for FIR filter prototype.
end
```

is used to take specific CallBack function action on the selected menu item.

See Also

ischeckd

zoomadd, zoomrep, zoomdel

Purpose

Add and delete lines from zoomtool.

Synopsis

```
zoomadd(axes,xdata,ydata)
zoomadd(axes,xdata,ydata,'current')

zoomrep(axes,ydata)
zoomrep(axes,xdata,ydata)

zoomdel(axes,'option')
```

Description

`zoomadd(axes,x,y)` adds line (*xdata,ydata*) to the zoomtool active in the axis pointed to with the handle *axes*.

`zoomadd(axes,xdata,ydata,'current')` makes the added line the zoomed line (current line with the cursors attached) in zoomtool.

`h = zoomadd(...)` returns a handle to the line added.

`zoomrep(axes,ydata)` replaces the zoomed line with *ydata* in the zoomtool active in the axes pointed to by the handle *axes*. The horizontal scaling remains the same. This form should be used when the overall length and scale of the zoomed line is unchanged.

`zoomrep(axes,xdata,ydata)` replaces the zoomed line with *ydata* using *xdata* for the horizontal scaling. This form should be used when the the zoomed line has changed lengths. For example, `zoomedit` uses this form after a cut or paste operation. The `zoomrep` function make low-level graphic calls to replace the zoomed line in zoomtool without destroying the cursors and other objects which may be contained in the data. There should only be the line in the zoomtool (see Toggle property of zoomtool). All the necessary changes are made to adapt zoomtool to the new line dependent upon the calling form of the function.

`zoomdel(axes,'option')` deletes lines from zoomtool active in the axes pointed to by the handle *axes* according to the specified 'option'.

Valid options are:

- 'current' - Deletes the line cursors are currently attached to.
- 'others' - Deletes all lines except the one the cursors are currently attached to.

Example

Create data for some lines. Note the *xdata* is sequential.

```
y1 = randn(100,1);
x2 = 51:150;
y2 = randn(100,1) + 3;
x3 = 101:250;
y3 = randn(150,1) + 6;
```

zoomadd

Plot the first line and add zoomtool.

```
plot(y1);
zoomtool
```

Add the *ydata* for the second line to zoomtool. Note the *xdata* must be the same as the *xdata* for the line currently in zoomtool.

```
xdata = get(zoomed(gcf),'XData');
zoomadd(gca,xdata,y2)
```

Delete the first line (the one the cursors still should be attached).

```
zoomdel(gca,'current')
```

Change the x-axis scale for the second line. Note that we have to supply the *ydata* to change the *xdata*.

```
zoomrep(gca,x2,y2)
```

Replace the *ydata* of the second line with that of the first without changing the x-axis scale.

```
zoomrep(gca,y1)
```

Replace the *zoomed* line with one of a different length.

```
zoomrep(gca,x3,y3)
```

See the use of *zoomadd* and *zoomdel* in the *spect2d* program. In *spect2d*, these functions are used to add and delete spectral estimates to the spectral plot.

See the use of *zoomrep* in the *zoomedit* function where it is used to replace the *zoomed* line after cut and paste operations.

See Also

zoomtool, *zoomrep*

INITIAL DISTRIBUTION LIST

- | | |
|---|---|
| 1. Defense Technical Information Center
Cameron Station
Alexandria, VA 22304-6145 | 2 |
| 2. Dudley Knox Library
Code 52
Naval Postgraduate School
Monterey, CA 93943-5101 | 2 |
| 3. Chairman, Code EC
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943 | 1 |
| 4. Dr. Monique P. Fargues, Code EC/Fa
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943 | 4 |
| 5. Dr. Herschel H. Loomis, Jr., Code EC/ Lm
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943 | 2 |
| 6. Dr. Charles W. Therrien, Code EC/ Ti
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943 | 1 |
| 7. LT Dennis W. Brown
Naval Information Warfare Activity
3801 Nebraska Ave. N.W.
Washington DC 20393 | 4 |
| 8. Dr. Cliff Comisky
NCCOSC RDTE Division, code 7701
San Diego, CA 92152-5000 | 1 |